# Design by Contract

Christos Dimoulas

## 1 Applying "Design by Contract"

```
@article{m:apply-design-by-contract,
author = {Meyer, Bertrand},
title = {Applying "Design by Contract"},
journal = {Computer},
volume = {25},
number = {10},
year = {1992},
pages = {40--51},
publisher = {IEEE Computer Society Press}}
```

**Content** The author introduces a new framework called design by contract for constructing robust components. Contracts are class invariants, pre-conditions and post-conditions. Class invariants correspond to class conistency constraints, pre-conditions to obligations of the component's user and post-conditions to obligations of the component's provider. Contracts are exposed at the interface of the components. The introduction of executable specification reduces in the code the need for complex defensive programming, encourages disciplined exception handling, improves documentation and helps understanding better the complexity of features like inheritance.

**Evaluation** The paper is ground-breaking. It proposes a powerful feature for light-weight program monitoring that also serves as the staring point for full-fledged program verification. It builds on previous work on program specification and verification and is the first presentation of a design framework for constructing reliable components from classes.

## 2 Design by Contract: The Lessons of Ariane

```
@article{jm:ariane
author = {J\'{e}z\'{e}quel, Jean-Marc and Meyer, Bertrand},
title = {Design by Contract: The Lessons of Ariane},
journal = {Computer},
volume = {30},
number = {1},
year = {1997},
pages = {129--130},
publisher = {IEEE Computer Society Press}}
```

**Content** The authors analyze the reasons for ESA's Ariane 5 launcher crash, a $500 disaster. The failure was caused by the re-use of a 10 year old component that raised an uncaught exception. The authors conclude that the failure could have been avoided if the re-used component was equipped with contracts that made explicit the component's requirements. Contracts could either monitor the execution and handle the exception or serve as the focus points for verification and testing techniques that could have revealed the bug.

**Evaluation** The analysis of the code production process is superficial. Why wasn't the dependence of Inertial Reference System on 16-bit arithmetic mentioned in its documentation if the manager, designer and implementor did their job correctly? However, the arguments about the importance of contracts for correct component re-use are valid and convincing. Despite the paper's weaknesses, the author makes clear that contracts can greatly improve code reliability.

## 3 Finding Implicit Contracts in .NET Libraries

```
@inproceedings{bm:find-dot-net-contracts,
author = {Karine Arnout Bertr and Bertrand Meyer},
title = {Finding Implicit Contracts in.NET Libraries},
booktitle = {Formal Methods for Components and Objects},
year = {2003},
pages = {258-318}}
```

**Content** The authors investigate if contracts are part of the integral process of writing code even in languages that do not offer support for design by contract. To this end, they analyze .NET libraries and discover that comments, purpose statements and exceptions hide class invariants, post-conditions and preconditions correspondingly. They manage to unearth a significant number of contracts from .NET's Collection library. They also discuss how the addition of contracts reveals subtleties of the implementation of this library. They propose improvements enabled by the addition of contracts that lead to more elegant and robust implementation and use of the library.

**Evaluation** The results of the paper are not surprising. In addition, the paper avoids to propose a systematic way to add a posteriori hidden contracts to a library's code. Especially, when the source code is not available (most common case) the paper seems to suggest that developers are better off rewriting the library from scratch and avoid reusing.

## 4 A Framework for Proving Contract-Equipped Classes

```
@inproceedings{m:contract-class-proof,
author = {Bertrand Meyer},
title = {A Framework for Proving Contract-Equipped Classes},
booktitle = {Abstract State Machines},
year = {2003},
pages = {108-125}}
```

**Content** The author identifies class correctness as contract satisfaction. In order to prove that the contracts hold, the author suggests introducing a mathematical abstract model that captures the class's properties as stated by its contracts. Also, an abstract class with contracts can be used to hide implementation details. The first step of the proof is to integrate the model and the abstract class and to show that the abstract class's contracts are consistent with the model's properties. After that, the proof focuses on lower implementation details and establishes that the code entails the model's properties.

**Evaluation** The idea of breaking up the proof in proof of consistency (interface) properties and proof of low level details mimicking the design of the code is interesting. However, this is analogous to what any good verification engineer would do by default.