35 Experiences with lightweight checks for mass-maintenance transformations

Niels Veerman
Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
nveerman@cs.vu.nl

Keywords automated modifications, lightweight approach, mass-maintenance

The use of automated modifications has gradually increased in the field of software maintenance. These maintenance transformations can be found where software needs to be maintained. Tools for code analysis have become more and more indispensable to maintain large systems, and large-scale modifications, such as Euro conversions, Y2K repairs or database migrations, have called for automated maintenance transformations, as argued for in [2].

The increasing interest in automated maintenance has heightened the need for control over automatic transformations. For instance, automated changes on business-critical systems should not jeopardize the operations of a company. Such changes usually consist of several complex transformations, making verification of an entire transformation on a large system difficult. Millions of lines of code can be affected by a mass update, and it is not very practical to inspect the resulting programs by hand. An alternative, extensive testing, can be expensive and not always feasible in practice. Automated massive changes are often carried out by software renovation companies, and they usually do not possess the required hard- and/or software to, for instance, compile and test a mainframe system. Although the owner of the system does have this option, compiling and testing can be an expensive exercise so possible errors should be detected in an early stage. Therefore, a different way is needed to control such maintenance transformations.

Another approach is to prove the correctness of a transformation in advance. However, to prove the correctness of mass-maintenance transformations, one would need the semantics of the programming language. Such an approach has several drawbacks. Besides the fact that there can be several different semantics for a language because of different dialects, compilers, compiler flags and operating systems (see [1]), it can be an expensive process and also prone to errors, and as soon as something changes (e.g. compiler version or operating system version) a different semantics needs to be developed. Moreover, one would also have to prove the used transformations system correct, for instance the preprocessor, parser and prettyprinter and so on. For these reasons, a correctness proof for mass-maintenance transformations on legacy systems is not practical, and we need other ways to make sure the updated systems are behaving as expected.

We have experimented with a lightweight approach to check large-scale maintenance transformations. Our starting point is two large-scale transformation projects that we carried out. Our approach concentrates on transformations for software maintenance, but can also be used in other areas. We present a range of checks to identify errors in transformations and discuss their application to real life cases.

Bibliography

[1] R. Lämmel and C. Verhoef. Cracking the 500-Language Problem. *IEEE Software*, pages 78–88, November/December 2001.

[2] C. Verhoef. Towards Automated Modification of Legacy Assets. *Annals of Software Engineering*, 9:315–336, 2000.