Generative software development has paved the way for the creation of multiple code generators and compilers that serve as a basis for automatically generating code to a broad range of software and hardware platforms. With full automatic code generation, the user is able to easily and rapidly synthetize software artifacts for various software platforms. In addition, modern generators (i.e., C compilers) become highly configurable, offering numerous configuration options that the user can use to easily customize the generated code for the target hardware platform.

In this context, it is crucial to verify the correct behaviour of code generators. Numerous approaches have been proposed to verify the functional outcome of generated code but few of them evaluate the non-functional properties of automatically generated code, namely the performance and resource usage properties.

This thesis addresses three problems:

**(1) Non-functional testing of code generators:** We benefit from the existence of multiple generators with comparable functionality (i.e., code generator families) to automatically test the generated code. We leverage the metamorphic testing to detect inconsistencies in code generators families by defining metamorphic relations as test oracles. We define the metamorphic relation as a comparison between the variations of performance and resource usage of code, generated from the same code generator family.

We evaluate our approach by analysing the performance of Haxe, a popular code generator family. Experimental results show that our approach is able to automatically detect several inconsistencies that reveal real issues in this family of code generators.

**(2) Compilers auto-tuning:** We exploit recent advances in search-based software engineering in order to provide an effective approach to tune compilers (e.g., GCC compilers) according to user's non-functional requirements (i.e., performance and resource usage). We also demonstrate that our approach can be used to automatically construct optimization levels that represent optimal trade-offs between multiple non-functional properties such as execution time and resource usage requirements.

**(3) Handling the diversity of software and hardware platforms in software testing:** Running tests and evaluating the resource usage in heterogeneous environments is tedious. To handle this problem, we benefit from the recent advances in lightweight system virtualization, in particular container-based virtualization, in order to offer effective support for automatically deploying, executing, and monitoring code in heterogeneous environment, and collect non-functional metrics (e.g., memory and CPU consumptions).