The history of software development shows a continuous increase of complexity in several aspects of the software development process. This complexity is highly correlated with the continuous innovation in software industry as more and more heterogeneous devices and software platforms are introduced in the market. As a consequence, generative programming techniques are increasingly applied in order to reduce the effort of software development and maintenance. Generative programming provides a software abstraction layer that software developers can use to express their needs and specify the desired system behavior (e.g., domain-specific languages, models, etc.). Simultaneously, 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.

Nevertheless, it is crucial that the software being automatically generated and configured by the user undergoes an appropriate testing technique to verify the correct generator behavior. 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) The problem of non-functional testing of code generators:** Testing code generators raises the oracle problem since there is no a clear definition of how the oracle might be defined when it comes to the test of the performance and resource usage properties. Therefore, we benefit from the existence of multiple generators with comparable functionality (i.e., code generator families) to automatically test the generated code. In fact, our proposed approach is based on the metamorphic testing paradigm to detect inconsistencies in code generators families by defining high-level test oracles (i.e., metamorphic relations). Experimental results show that our approach is able to automatically detect some performance inconsistencies that reveal real issues in existing code generator families.

**(2) The problem of compilers auto-tuning:** Auto-tuning compilers is complex because of the large possible optimizations that can be applied. Therefore, we benefit from recent advances in search-based software engineering in order to provide an effective approach to explore the large optimization search space and automatically tuning 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) The problem of software platforms diversity and hardware heterogeneity in software testing:** Running tests and evaluating the resource usage metrics 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 of generated (or not) code in heterogeneous environment, based on containers.