Program of Studies:	**Master Program Bioinformatics**
Name of the module:	**Software Engineering**
Abbreviation:	**I-M-10**
Subtitle:	Core Lecture
Modules:	Lecture: 4 h (weekly) Tutorial: 2 h (weekly)
Semester:	1st -3rd semester/at least every two years
Responsible lecturer:	Prof. Dr. Sven Apel
Lecturer:	Prof. Dr. Sven Apel
Language:	English
Level of the unit/ Mandatory or not:	Graduate course / mandatory elective
Total workload:	270 h = 90 h of classes and 180 h private study;
Credits:	9
Entrance requirements:	- Knowledge of programming concepts (as taught in the lectures *Programmierung 1* and *Programmierung 2*) - Basic knowledge of software processes, design, and testing (as taught and applied in the lecture *Softwarepraktikum*)
Aims/Competences to be developed:	- The students know and apply modern software development techniques. - They are aware of key factors contributing to the complexity of real-world software systems, in particular, software variability, configurability, feature interaction, crosscutting concerns, and how to address them. - They know how to apply established design and implementation techniques to master software complexity. - They are aware of advanced design and implementation techniques, including collaboration-based design, mixins/traits, aspects, pointcuts, advice. - They are aware of advanced quality assurance techniques that take the complexity of real-world software systems into account: variability-aware analysis, sampling, feature-interaction detection, predictive performance modeling, etc. - They appreciate the role of non-functional properties and know how to predict and optimize software systems regarding these properties. - They are able to use formal methods to reason about key techniques and properties covered in the lecture.

Content:	- Domain analysis, feature modeling - Automated reasoning about software configuration using SAT solvers 36 - Runtime parameters, design patterns, frameworks - Version control, build systems, preprocessors - Collaboration-based design - Aspects, pointcuts, advice - Expression problem, preplanning problem, code scattering & tangling, tyranny of the dominant decomposition, inheritance vs. delegation vs. mixin composition - Feature interaction problem (structural, control- & data-flow, behavioral, non-functional feature interactions) - Variability-aware analysis and variational program representation (with applications to type checking and static program analysis) - Sampling (random, coverage) - Machine learning for software performance prediction and optimization
Assessment/Exams:	Beside the lecture and weekly practical exercises, there will be a number of assignments in the form of mini-projects for each student to work on (every two to three weeks). The assignments will be assessed based on the principles covered in the lecture. Passing all assignments is a prerequisite for taking the final written exam. The final grade is determined only by the written exam. Further examination details will be announced by the lecturer at the beginning of the course. In short: - Passing all assignments (prerequisite for the written exam) - Passing the written exam
Grade:	The grade is determined by the written exam. Passing all assignments is a prerequisite for taking the written exam. The assignments do not contribute to the final grade. Further examination details will be announced by the lecturer at the beginning of the course.
Literature:	- Feature-Oriented Software Product Lines: Concepts and Implementation. S. Apel, et al., Springer, 2013. - Generative Programming: Methods, Tools, and Applications: Methods, Techniques and Applications. K. Czarnecki, et al., Addison-Wesley, 2000. - Mastering Software Variability with FeatureIDE. J. Meinicke, et al., Springer, 2017.