Studiengang:	Bachelor Bioinformatik
Modulbezeichnung:	**Software Engineering Lab**
ggf. Kürzel:	PI-B-1
ggf. Lehrveranstaltungen:	Daily exercises and lectures (first few weeks) Daily project work with tutoring
Semester:	2. Semester
Angebotsturnus:	Lecture free time after ss
Modulverantwortliche(r):	Prof. Dr. Sven Apel
Dozent(in):	Prof. Dr. Sven Apel Dr. Norman Peitek
Sprache:	Englisch
Zuordnung zum Curriculum:	Pflichtmodulelement der Kategorie „Praktikum der Informatik“
Lehrform / SWS:	Block (7 weeks)
Arbeitsaufwand:	270 h = 35 h of lectures and exercises + 235 h project work
Kreditpunkte:	9
Voraussetzungen:	Participation in the Software Engineering lab requires extensive programming skills as taught in the courses Programming 1 and Programming 2. A passing grade in Programming 2 is required to enroll in this course. Students are required to bring their own laptops
Lernziele / Kompetenzen:	Participants acquire the ability to solve complex software development problems individually and in teams. Participants are aware of common problems and pitfalls of software development and know how to address them. Participants are able to accomplish and coordinate software development tasks based on a set of given requirements. For this purpose, they are able to select proper methods and techniques to minimize risks and maximize software quality. Participants know about foundations and principles of software design, including cohesion, coupling, modularity, encapsulation, abstraction, and information hiding. They are acquainted with a whole array of design patterns, knowing their aim and individual strengths and weaknesses. They are able to apply design patterns beneficially and to judge and improve the quality of software designs. Participants master fundamental techniques and tools for software testing, debugging, and version control.

Inhalt:	• Software design • Software testing • Team work • Debugging
Studien-Prüfungsleistungen:	The goal of the Software Engineering Lab is to develop a non-trivial software system in a team effort. In this course, a number of documents (design models, documentation, implementation plan etc.) and artifacts (source code, tests, etc.) need to be developed and submitted. Correctness, completeness, quality, and timely submission of all documents and artifacts are major grading criteria. The Software Engineering Lab consists of two phases: exercise phase and group phase. In the exercise phase, participants will complete an entry exam, covering current topics from the lecture. Only participants that have passed the exercise phase will be admitted tot he group phase. In the group phase, participants will first design and then implement, and test a substantial software system in a team effort, an dsubmit both their design and their implementation (including tests) for evaluation. All documents (design models, documentation, implementation plan etc.) and artifacts (source code, tests, etc.) of the group phase will be evaluated based on the principles and quality criteria conveyed in the lectures. To pass the group phase, students must pass both the design submission and the implementation submission, and prove individually their substantial contribution to the group project. More details on the exams will be announced at the beginning of the course.
Literatur:	• Software Engineering. I. Sommerville, Addison-Wesley, 2004. • Software Engineering: A Practioner's Approach. R. Pressman, McGraw Hill Text, 2001. • Using UML: Software Engineering with Objects and Components. P. Stevens, et al., Addison-Wesley, 1999. • UML Distilled. M. Fowler, et al., Addison-Wesley, 2000. • Objects, Components and Frameworks with UML, D. D'Souza, et al., Addison-Wesley, 1999. • Designing Object-Oriented Software. R. Wirfs-Brock, et al., Prentice Hall, 1990. • Design Patterns. Elements of Reusable Object-Oriented Software. E. Gamma, et al., Addison-Wesley, 1995. • Head First Design Patterns. E. Freeman, et al. O'Reilly, 2004. • Software Architecture: Perspectives on an Emerging Discipline. M. Shaw, et al., Prentice-Hall, 1996. • Refactoring: Improving the Design of Existing Code. M. Fowler, et al., Addison-Wesley, 1999. • Software Testing and Analysis: Process, Principles and Techniques. M. Pezze, Wiley. 2007.