To reason about a program means to state or conclude, by logical means, some properties the program exhibits; like its correctness according to certain expected behavior. The continuous need for more ambitious, more complex, and more dependable software systems demands for better mechanisms to modularize them and reason about their correctness. The reasoning process is affected by the design decisions made by the developer of the program and by the features supported by the programming language used. Beyond Object Orientation, Implicit Invocation and Aspect Oriented languages pose very hard reasoning challenges. Important tradeoffs must be considered while reasoning about a program: modular vs. non-modular reasoning, case-by-case analysis vs. abstraction, explicitness vs. implicitness; are some of them. By deciding a series of tradeoffs one can configure a reasoning scenario. For example if one decides for modular reasoning and explicit invocation a well known object oriented reasoning scenario can be used. This dissertation identifies various important tradeoffs faced when reasoning about implicit invocation and aspect oriented programs, characterize scenarios derived from making choices regarding these tradeoffs, and provides sound proof rules for verification of programs covered by all these scenarios. Guidance for program developers and language designers is also given, so that reasoning about these types of programs becomes more tractable.