Code-breaking algorithms have also existed for centuries. The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. Programs were mostly entered using punched cards or paper tape. The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. Code-breaking algorithms have also existed for centuries. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. As early as the 9th century, a programmable music sequencer was invented by the Persian Banu Musa brothers, who described an automated mechanical flute player in the Book of Ingenious Devices. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. Auxiliary tasks accompanying and related to programming include analyzing requirements, testing, debugging (investigating and fixing problems), implementation of build systems, and management of derived artifacts, such as programs' machine code. Use of a static code analysis tool can help detect some possible problems. For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash. Many programmers use forms of Agile software development where the various stages of formal software development are more integrated together into short cycles that take a few weeks rather than years. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se. Scripting and breakpointing is also part of this process. Use of a static code analysis tool can help detect some possible problems.