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. 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. Ideally, the programming language best suited for the task at hand will be selected. 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. However, readability is more than just programming style. Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. Proficient programming usually requires expertise in several different subjects, including knowledge of the application domain, details of programming languages and generic code libraries, specialized algorithms, and formal logic. Ideally, the programming language best suited for the task at hand will be selected. Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Ideally, the programming language best suited for the task at hand will be selected. 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. Programs were mostly entered using punched cards or paper tape. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Programs were mostly entered using punched cards or paper tape. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. 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. 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. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. It is usually easier to code in "high-level" languages than in "low-level" ones. It is usually easier to code in "high-level" languages than in "low-level" ones.