# Chapter 4: Software Process

The capabilities of software have, no doubt, changed dramatically since the introduction of computers decades ago. Society’s increasing reliance on computers and software creates pressure for late-adopters to take the risk of using software. Software evolves because individuals and organizations are integrating computers into their activities.

In order to build quality software systems, many organizations follow a formal approach to the software development process including formal methods, best practices, and adoption of standards. Many models have already been created and are universally understood as the “state-of-the-art” in software engineering [cite:waterfall/spiral/etc]. But the primary feature that these models have in common is *iteration*.

## 4.1 Reasonably Prudent Software Development

salient point in *all* models is iteration – design/implement/test, repeat!

## 4.2 The Software Engineering State-of-the-Art

The state-of-the-art in software engineering refers to the “latest-and-greatest” tools, methodologies, and practices as agreed upon by members of the industry.

### 4.2.1 Output

The state-of-the-art is created by software engineers themselves! Industry specialists or academic researchers come up with ideas that help projects they are working on and publish them. The work is reviewed; sometimes by a jury of peers, sometimes by the industry, sometimes by the users of the work. After several iterations of revisions, the idea garners the support of many and is accepted by a wide user base. As such, the industry and its members establish the work as state-of-the-art.

Many examples of the creation of software engineering state-of-the-art. Consider the C compiler. Researchers from AT&T Bell Labs sought a way to simplify programming for microprocessors; something to make code more readable for humans. Before language compilers existed, programmers would use machine assembly to program computers. Programmers wrote the exact statements that were interpreted and executed by the processor.

Dennis Ritchie created a set of basic control statements and a syntax that would become the C programming language. The same program written in assembly code could be rewritten in C in far fewer lines and humans would have an easier time understanding the program. The compiler would convert the high-level programming language into a machine assembly that a computer could understand.

|  |  |
| --- | --- |
| int main(char\* args)  {  int counter = 0;  int i;  for (i = 0; i < 10; i++)  {  counter += i;  }  } | .text  .globl \_main  \_main:  pushl %ebp  movl %esp, %ebp  subl $24, %esp  movl $0, -16(%ebp)  movl $0, -12(%ebp)  jmp L2  L3:  movl -12(%ebp), %eax  leal -16(%ebp), %edx  addl %eax, (%edx)  leal -12(%ebp), %eax  incl (%eax)  L2:  cmpl $9, -12(%ebp)  jle L3  leave  ret  .subsections\_via\_symbols |

The two code fragments above do the exact same thing: compute the sum of all integers up to 10. The code on the left is written in C, while the code on the right is written in assembly.

But the C compiler did not immediately earn the attention or credit that it would later have. Initial iterations of the compiler would have for sure been buggy and engineers would be reluctant to use it. In fact, some organizations even forbade the use of high-level language compilers and forced their programmers to use a trusted assembly language instead.

### 4.2.2 Input

When a software project begins, the team of software engineers decides what tools are best suited for the work they are to do. Likely, the team will employ the technologies that are best suited for the project and what the industry considers state-of-the-art.

Consider Microsoft’s Mediaroom IPTV platform. Engineers decided to build the set-top-box client on top of the .NET compact framework. The .NET CF was chosen as a platform to build on because the Microsoft engineers consider it state-of-the-art and the most appropriate technology for the job. If problems are found, developers can report them to the engineers that work on the .NET platform.

For example, the .NET compact framework lacks useful try-parse methods to convert strings to primitive types like integers and booleans. While this feature exists for the PC version of the framework, Mediaroom engineers filed a feature request for this useful pattern to be included in the compact version. Framework engineers can revise the platform and state-of-the-art is updated.

This pattern of updating the state-of-the-art through usage is especially apparent in open source software like the Linux kernel or the many projects built by the Apache Software Foundation.

# Chapter 5: Paving the Intersection