Computer Programming as an Art

by Donald E. Knuth

1974. ACM Turing Award Lecture

[The Turing Award citation read by Bernard A.Galler, chairman of the 1974 Turing Award Committee, on the presentation of this lecture on November 11 at the ACM Annual Conference in SanDiego.]

> 1974年のチューリング賞委員会の議長であるバーナードA.ガラーが11月11日にサンディエゴで開催されたACM年次会議でのこの講演のプレゼンテーションに関するチューリング賞の引用。

The A.M. Turing Award of the ACM is presented annually by the ACM to an individual selected for his contributions of a technical nature made to the computing community. In particular, these contributions should have had significant influence on a major segment of the computer field.

> 午前ACMのチューリング賞は、コンピューティングコミュニティに対する技術的性質の貢献に対して選ばれた個人にACMによって毎年贈られます。特に、これらの貢献はコンピュータ分野の主要なセグメントに大きな影響を与えたはずです。

"The 1974 A.M. Turing Award is presented to Professor Donald E. Knuth of Stanford University for a number of major contributions to the analysis of algorithms and the design of programming languages, and in particular for his most significant contributions to the 'art of computer programming' through his series of well-known books.

> 西暦1974年、スタンフォード大学のドナルド E. クヌース教授に、アルゴリズムの分析とプログラミング言語の設計への数々の主要な貢献に対して、また、特に教授の最も重要な貢献として彼による有名な本のシリーズである「Art of computer programming」に対してチューリング賞が授与されました。

The collections of techniques, algorithms and relevant theory in these books have served as a focal point for developing curricula and as an organizing influence on computer science."

> これらの本に記載された技術、アルゴリズムそして関連する理論のコレクションはカリキュラムを開発するための焦点として、また、コンピュータサイエンスにおける影響？を体系的にまとめる事に役に立ちます。

Such a formal statement cannot put into proper perspective the role which Don Knuth has been playing in computer science, and in the computer industry as a whole.

> そのような正式な声明はクヌースがコンピューターサイエンスおよびコンピューター業界全体で果たしてきた役割を適切な考え方に入れることができません。

It has been my experience with respect to the first recipient of the Turing Award, Professor Alan J. Perlis, that at every meeting in which he participates he manages to provide the insight into the problems being discussed that becomes the focal point of discussion for the rest of the meeting.

> チューリング賞の最初の受賞者であるアランJ.ペルリス教授に関する私の経験は、彼が参加するすべての会議で彼は議論されている問題への洞察を提供するために管理しますそれが会議の残りの議論の焦点になります。

In a very similar way, the vocabulary, the examples, the algorithms, and the insight that Don Knuth has provided in his excellent collection of books and papers have begun to find their way into a great many discussions in almost every area of the field.

> 非常に似た方法で、語彙、例、アルゴリズム、およびDon Knuthが提供した洞察 彼の優れた本や論文のコレクションは見つけ始めましたほぼすべての分野で非常に多くの議論への道フィールド。

This does not happen easily.

> これは簡単には起こりません。

As every author knows, even a single volume requires a great deal of careful organization and hard work.

> すべての著者が知っているように、1つのボリュームであっても、多くの注意深い整理と努力が必要です。

All the more must we appreciate the clear view and the patience and energy which Knuth must have had to plan seven volumes and to set about implementing his plan so carefully and thoroughly.

> さらに、Knuthが7巻を計画し、彼の計画を非常に慎重かつ徹底的に実行するために持っていたはずの明確な見解と忍耐とエネルギーを高く評価する必要があります。

It is significant that this award and the others that he has been receiving are being given to him after three volumes of his work have been published.

> この賞と彼が持っている他の賞が重要です彼の3巻の後に受信されている彼に与えられています作品が公開されています。

We are clearly ready to signal to everyone our appreciation of Don Knuth for his dedication and his contributions to our discipline.

> ドンクヌースの献身と私たちの規律への貢献に対する感謝を、誰にでもはっきりと伝える準備ができています。

I am very pleased to have chaired the Committee that has chosen Don Knuth to receive the 1974 A.M. Turing Award of the ACM.

> 私は、1974年のチューリング賞の授与にドンクヌースを選んだ委員会の議長を務めたことを非常に嬉しく思います。

When Communications of the ACM began publication in 1959, the members of ACM'S Editorial Board made the following remark as they described the purposes of ACM's periodicals:

> ACMの通信が1959年に出版を開始したとき、ACMの定期刊行物の目的を説明するため、ACMの編集委員会のメンバーは次のように述べました。

"If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected."

> コンピュータープログラミングがコンピューターの研究開発の重要な部分となるためには、プログラミングを芸術から規律ある科学に移行させる必要があります。

Such a goal has been a continually recurring theme during the ensuing years; for example, we read in 1970 of the "first steps toward transforming the art of programming into a science".

> このような目標は、その後数年間、継続的に繰り返される、例えば、1970年の「プログラミングの技術を科学に変えるための最初のステップ」といった、テーマでした。

Meanwhile we have actually succeeded in making our discipline a science, and in a remarkably simple way: merely by deciding to call it "computer science".

> 一方、私たちは、自分の専門分野を科学にすることに実際に成功し、それを単に「コンピューターサイエンス」と呼ぶことにしました。

Implicit in these remarks is the notion that there is something undesirable about an area of human activity that is classified as an "art"; it has to be a Science before it has any real stature.

> これらの発言に暗示されているのは、「芸術」として分類される人間の活動の領域に関して望ましくない何かがあるという概念です。それは本当の身長を持つ前に科学でなければなりません。

On the other hand, I have been working for more than 12 years on a series of books called "The Art of Computer Programming."

> 一方、私は「コンピュータプログラミングの芸術」と呼ばれる一連の本に12年以上取り組んでいます。

People frequently ask me why I picked such a title; and in fact some people apparently don't believe that I really did so, since I've seen at least one bibliographic reference to some books called "The Act of Computer Programming."

> なぜ私がそのようなタイトルを選んだのかとよく聞かれます。実際、「コンピュータプログラミングの行為」と呼ばれるいくつかの本の書誌参照を少なくとも1つは見たので、実際にそうしたとは思わない人もいます。

In this talk I shall try to explain why I think "Art" is the appropriate word.

> この講演では、「アート」が適切な言葉だと思う理由を説明しようと思います。

I will discuss what it means for something to be an art, in contrast to being a science; I will try to examine whether arts are good things or bad things; and I will try to show that a proper viewpoint of the subject will help us all to improve the quality of what we are now doing.

> 科学であることとは対照的に、何かが芸術であることの意味を説明します。芸術が良いものか悪いものかを調べてみます。そして、主題の適切な視点が、私たち全員が私たちが今していることの質を改善するのに役立つことを示すようにします。

One of the first times I was ever asked about the title of my books was in 1966, during the last previous ACM national meeting held in Southern California.

> 私の本のタイトルについて初めて尋ねられたのは、南カリフォルニアで開催された前回のACM全国会議の1966年でした。

This was before any of the books were published, and I recall having lunch with a friend at the convention hotel.

> これは本が出版される前のことで、コンベンションホテルで友人と昼食をとったことを思い出します。

He knew how conceited t was, already at that time, so he asked if I was going to call my books "An Introduction to Don Knuth".

> 彼はすでにその時点でどれほどうぬぼれているかを知っていたので、私の本を「ドン・クヌースの紹介」と呼ぶかどうか尋ねました。

I replied that, on tile contrary, I was naming the books after him.

> それに反して、私は彼にちなんで本に名前を付けていたと答えた。

His name: Art Evans. (The Art of Computer Programming, in person.)

> 彼の名前：アートエヴァンス。（コンピュータープログラミングの技術、直接。）

From this story we can conclude that the word "art" has more than one meaning.

> この物語から、「芸術」という言葉には複数の意味があると結論付けることができます。

In fact, one of the nicest things about the word is that it is used in rnany different senses, each of which is quite appropriate in connection with computer programming.

> 実際、この単語の最も素晴らしい点の1つは、あらゆる言語で使用されていることです。 それぞれがコンピュータープログラミングに関連して非常に適切です。

While preparing this talk, I went to the library to find out what people have written about tire word "art" through the years; and after spending several fascinating days in the stacks, I came to the conclusion that "art" must be one of the most interesting words in the English language.

> この講演の準備をしている間に、私は図書館に行って、人々が長年にわたって「芸術」という言葉について書いたことを見つけました。そして、スタックで数日間魅力的な日を過ごした後、私は「芸術」が英語で最も興味深い言葉の1つでなければならないという結論に達しました。

The Arts of Old

If we go back to Latin roots, we find ars, artis meaning "skill." It is perhaps significant that the corresponding Greek word was *****, the root of both "technology" and "technique."

> ラテンのルーツをたどると、「芸術」という言葉は「技術」を意味します。 対応するギリシャ語が「テクノロジー」と「テクニック」両方のルーツになる＊＊＊＊であることはおそらく重要です。

Nowadays when someone speaks of "art" you probably think first of "fine arts" such as painting and sculpture, but before the twentieth century the word was generally used in quite a different sense. Since this older meaning of "art" still survives in many idioms, especially when we are

contrasting art with science, I would like to spend the next few minutes talking about art in its classical sense.

> 今日では、「art」というと、おそらく最初に思い浮かべるのは、絵画や彫刻といった「美術」ですが、20世紀以前は一般的に完全に違う状況で使われていました。 多くの慣用句、特に科学と芸術を比較する際に、この「art」の古い意味はまだ有効なので、少し古典的な意味で、「art」について話したいと思います。

In medieval times, the first universities were established to teach the seven so-called "liberal arts," namely grammar, rhetoric, logic, arithmetic, geometry, music, and astronomy. Note that this is quite different from the curriculum of today's liberal arts colleges, and that at least three of the original seven liberal arts are important components of computer science. At that time, an "art" meant something devised by man's intellect, as opposed to activities derived from nature or instinct; "liberal arts" were liberated or free, in contrast to manual arts such as plowing. During the middle ages the word "art" by itself usually meant logic , which usu.ally meant the study of syllogisms.

> 中世において、最初の大学はいわゆる７つの教養科目、すなわち、文法、文学、論理学、算術、幾何学、音楽、天文学といったものを教えるために設立されました。 これは、現在の大学の教養科目のカリキュラムとは全く異なり、元の7つの教養科目のうち、少なくとも３つはコンピュータサイエンスの要素として重要であることに注目してください。 当時、「芸術」とは、自然や本能に由来する活動とは対照的に、人間の知性によって考案されたものを意味していました。教養科目は、掘るといった手作業の芸術とは対照的に、解放されたか自由でした。

Science vs. Art

The word "science" seems to have been used for many years in about the same sense as "art"; for example, people spoke also of the seven liberal sciences, which were the same as the seven liberal arts. Duns Scotus in the thirteenth century called logic "the Science of Sciences, and the Art of Arts". As civilization and learning developed, the words took on more and more independent meanings, "science" being used to stand for knowledge, and "art" for the application of knowledge. Thus, the science of astronomy was the basis for the art of navigation. The situation was almost exactly like the way in which we now distinguish between "science" and "engineering."

> 「科学」という言葉は、「芸術」とほぼ同じ意味で長年使用されているようです。たとえば、人々はまた、7つのリベラルサイエンスについて話しました、これは、7つの教養科目と同じでした。13世紀のDuns Scotusは、論理を「科学の中の科学と芸術の中の芸術」と呼びました。 文明と学問が発展するにつれ、その言葉はますます独立した意味を帯びるようになり、科学は知識を表すために使用され、「芸術」は知識の応用のために使われました。 したがって、天文学の科学は航海術の基礎でした。この状況は、「科学」と「工学」を区別する方法とほぼ同じでした。

Many authors wrote about the relationship between art and science in the nineteenth century, and I believe the best discussion was given by John Stttart Mill. He said the following things, among others, in 1843:

> 多くの著者が19世紀の芸術と科学の関係について書いており、最高の議論はJohn Stttart Millによって与えられたと思います。彼は1843年に、とりわけ以下のことを言った：

> Several sciences are often necessary to from the groundwork of a single art. Such is the complication of human affairs, that to enable one thing to be done, it is often requisite to know the nature and properties of many things ... Art in general consists of the truths of Science, arranged in the most convenient order for practice, instead of the order which is the most convenient for thought. Science groups and arranges its truths so as to enable us to take in at one view as much as possible of the general order of the universe. Art brings together from parts of the field of science most remote from one another, the truths relating to the production of the different and heterogeneous conditions necessary to each effect which the exigencies of practical life require.

> 一つの芸術の基礎を整えるには、しばしばいくつかの科学が必要です。 これは人間の仕事の複雑さであり、1つのことができるようになるには、本質や多くのものの特性を知ることがしばしば必要です。一般に、芸術は科学の真理から成り、思考にとって最も便利な順序ではなく、実践にとって最も便利な順序で配置されています。科学は、宇宙の一般的な秩序を可能な限り一目で把握できるように、その真実をグループ化し、整理します。芸術は、互いに最も離れた科学分野の一部、実生活の緊急性が必要とする各効果に必要な異なる異質な条件の生成に関する真実を結び付けます。

As I was looking up these things about the meanings of "art," I found that authors have been calling for a transition from art to science for at least two centuries. For example, the preface to a textbook on mineralogy, written in 1784, said the following: "Previous to the year 1780, mineralogy, though tolerably understood by many as an Art, could scarce be deemed a Science."

> 「芸術」の意味についてこれらのことを調べていると、著者は少なくとも2世紀の間、芸術から科学への移行を求めてきたことがわかりました。 たとえば、1784年に書かれた鉱物学の教科書の序文は次のように述べています。「鉱物学は、多くの人が芸術として容認できるものの、1780年以前は科学と見なされることはほとんどありませんでした。」

According to most dictionaries "science" means knowledge that has been logically arranged and systematized in the form of general "laws." The advantage of science is that it saves us from the need to think things through in each individual case; we can turn our thoughts to higher-level concepts. As John Ruskin wrote in 1853: "The work of science is to substitute facts for appearances, and demonstrations tbr impressions."

> ほとんどの辞書によると、「科学」とは、一般的な「法律」の形で論理的に整理され体系化された知識を意味します。 科学の利点は、個々のケースごとに物事を考える必要性から私たちを救うことです。考えをより高いレベルの概念に変えることができます。 ジョン・ラスキンが1853年に書いたように、「科学の仕事は外見を事実に置き換え、印象をデモンストレーションすることです」。

It seems to me that if the authors I studied were writing today, they would agree with the following characterization: Science is knowledge which we understand so welt that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it. Since the notion of an algorithm or a computer program provides us with an extremely useful test for the depth of our knowledge about any given subject, the process of going from an art to a science means that we learn how to automate something.

> 私が研究した著者が今日執筆している場合、彼らは次の特徴づけに同意するように思えます。科学は私たちが理解している知識なので、コンピューターに教えることができるでしょう。そしてもし私たちが何かを完全に理解していなければ、それを扱うのは芸術です。アルゴリズムまたはコンピュータ

> ープログラムの概念は、特定の主題に関する知識の深さについて非常に有用なテストを提供するため、芸術から科学へと進むプロセスは、何かを自動化する方法を学ぶことを意味します。

Artificial intelligence has been making significant progress, yet there is a huge gap between what computers can do in the foreseeable future and what ordinary people can do. The mysterious insights that people have when speaking, listening, creating, and even when they are programming, are still beyond the reach of science; nearly everything we do is still an art.

> 人工知能は大きな進歩を遂げていますが、近い将来にコンピューターができることと、 普通の人ができます。人々が話したり、聞いたり、作成したり、プログラミングを行ったりするときに持つ不思議な洞察は、科学の範囲を超えています。私たちが行うほとんどすべてのことはまだ芸術です。

From this standpoint it is cer[ainly desirable to make computer programming a science, and we have indeed come a long way in the 15 years since the publication of the remarks I quoted at the beginning of this talk. Fifteen years ago computer programming was so badly understood that hardly anyone even thought about proving programs correct; we just fiddled with a program until we "knew" it worked. At that time we didn't even know how to express the concept that a program was correct, in any rigorous way. It is only in recent years that we have been learning about the processes of abstraction by which programs are written and understood; and this new knowledge about programming is currently producing great payoffs in practice, even though few programs are actually proved correct with complete rigor, since we are beginning to understand the principles of program structure. The point is that when we write programs today, we know that we could in principle construct formal proofs of their correctness if we really wanted to, now that we understand how such proofs are formulated. This scientific basis is resulting in programs that are significantly more reliable than those we wrote in former days when intuition was the only basis of correctness.

> この観点から、コンピュータープログラミングを科学にすることは非常に望ましいことであり、この講演の冒頭で引用した発言が発表されてから15年で、私たちは確かに大きな進歩を遂げました。15年前、コンピュータープログラミングは非常によく理解されていなかったため、プログラムの正しさを証明しようと考える人はほとんどいませんでした。プログラムが機能することが「わかる」までプログラムをいじりました。 当時、私たちは、プログラムがどんな厳密な方法においても正しいという概念を表現する方法すら知りませんでした。最近になってようやく、私たちはプログラムを作成および理解するための抽象化のプロセスについて学びました。プログラム構造の原則を理解し始めているため、完全な厳密さで実際に正しいと証明されたプログラムはほとんどありませんが、プログラミングに関するこの新しい知識は現在、実際に大きな見返りを生み出しています。 要点は、今日プログラムを書くとき、そのような証明がどのように定式化されるかを理解したので、本当にしたいなら、原則としてそれらの正確さの形式的な証明を構築できることを知っているということです。この科学的根拠は、直観が正確さの唯一の根拠であった以前に書いたものよりもはるかに信頼性の高いプログラムをもたらしています。

The field of "automatic programming" is one of the major areas of artificial intelligence research today. Its proponents would love to be able to give a lecture entitled "Computer Programming as an Artifact" (meaning that programming has become merely a relic of bygone days), because their aim is to create machines that write programs better than we can, given only the problem specification. Personally t don't think such a goal will ever be completely attained, but I do think that their research is extremely important, because everything we learn about programming helps

us to improve our own artistry, in this sense we should continually be striving to transform every art into a science: in the process, we advance the art.

> 「自動プログラミング」の分野は、今日の人工知能研究の主要な分野の1つです。その提案者は、「アーティファクトとしてのコンピュータープログラミング」というタイトルの講義を提供できることを望んでいます（つまり、プログラミングは単なる遺物になったということです）。 なぜなら、彼らの目的は、問題の仕様だけを考えて、私たちができる以上にプログラムを書くマシンを作ることだからです。

Science and Art

Our discussion indicates that computer programming is by now both a science and an art, and that the two aspects nicely complement each other. Apparently most authors who examine such a question come to this same conclusion, that their subject is both a science and an art, whatever their subject is "The Science of Playwriting". I found a book about elementary photography, written in 1893, which stated that "the development of the photographic image is both an art and a science". In fact, when I first picked up a dictionary in order to study the words "art" and "science," I happened to glance at the editor's preface, which began by saying, "The making of a dictionary is both a science and an art." The editor of Funk & Wagnall's dictionary observed that the painstaking accumulation and classification of data about words has a scientific character, while a wellchosen phrasing of definitions demands the ability to write with economy and precision: "The science without tile art is likely to be neffective; the art without the science is certain to be inaccurate."

> 私たちの議論は、コンピュータープログラミングが今では科学と芸術の両方であり、2つの側面が互いにうまく補完し合っていることを示しています。どうやらそのような質問を検討するほとんどの著者は、彼らの主題が「プレイライティングの科学」であっても、科学と芸術の両方であるという同じ結論に達します。私は、1893年に書かれた「写真画像の開発は芸術でもあり科学でもある」と書かれた小学校写真に関する本を見つけました。実際、「芸術」という言葉を勉強するために辞書を最初に取り上げたとき、そして「科学」、私はたまたま「辞書を作ることは科学であり芸術である」と言って編集者の序文を一目見た。 Funk＆Wagnallの辞書の編集者は、言葉に関するデータの骨の折れる蓄積と分類には科学的特徴があり、定義の適切なフレージングは経済と正確さで書く能力を要求することを観察しました。 科学のない芸術は不正確であると確信しています。

When preparing this talk I looked through the card catalog at Stanford library to see how other people have been using tile words "art" and "science" in the titles of their books. This turned out to be quite interesting.

> この講演の準備をするとき、私はスタンフォード図書館のカードカタログを見て、他の人が本のタイトルで「芸術」と「科学」という言葉をどのように使用しているかを調べました。これは非常に興味深いことが判明しました。

For example, I found two books entitled The Art of Playing the Piano, and others called The Science of Pianoforte Technique, The Science of Pianoforte Practice. There is also a book called The Art of Piano Playing: A Scientific Approach.

> たとえば、「ピアノを弾く芸術」というタイトルの2冊の本と、ピアノフォルテ技術の科学、ピアノフォルテ練習の科学という本を見つけました。「ピアノ演奏の芸術：科学的アプローチ」という本もあります。

Then I found a nice little book entitled The Gentle Art cf Mathematics, which made me somewhat sad that I can't honestly describe computer programming as a "gentle art".

> それから、数学を参照したThe Gentle Artというタイトルの素敵な小さな本を見つけました。それは、コンピュータプログラミングを「穏やかな芸術」として正直に説明できないことを少し悲しくさせました。

I had known for several years about a book called The Art of Computation, published in San Francisco, 1879, by a man named C. Frusher Howard. This was a book on practical business arithmetic that had sold over 400,000 copies in various editions by 1890. I was amused to read the preface, since it shows that Howard's philosophy and the intent of his title were quite different from mine; he wrote: "A knowledge of the Science of Number is of minor importance; skill in the Art of Reckoning is absolutely indispensible."

> 1879年にサンフランシスコで出版されたC. Frusher Howardという男のThe Art of Computationという本について、私は数年前から知っていました。 これは、1890年までにさまざまなエディションで40万部以上を販売していた実用的なビジネス算術の本でした。ハワードの哲学と彼のタイトルの意図が私のものとはかなり異なっていたことを示しているので、私は序文を読んで面白かった。彼は次のように書いた：「数の科学の知識はあまり重要ではない。レコニングの技術は絶対に不可欠である。」

Several books mention both science and art in their titles, notably The Science of Being and Art of Living by Maharishi Mahesh Yogi. There is also a book called The Art of Scientific Discovery, which analyzes how some of the great discoveries of science were made.

> 数冊の本では、タイトルに科学と芸術の両方、特にマハリシ・マヘシュ・ヨギによる存在の科学と生活の芸術に言及しています。科学の発見の芸術と呼ばれる本もあります。それは、科学の偉大な発見のいくつかがどのようになされたかを分析します。

So much for the word "art" in its classical meaning. Actually when I chose the title of my books, I wasn't thinking primarily of art in this sense, I was thinking more of its current connotations. Probably the most interesting book which turned up in nay search was a fairly recent work by Robert E. Mueller called The Science of Art. Of all the books I've mentioned, Mueller's comes closest to expressing what I want to make the central theme of my talk today, in terms of real artistry as we now understand the term. He observes: "It was once thought that the imaginative outlook of the artist was death for' the scientist. And the logic of science seemed to spell 'doom to all possible artistic flights of fancy." He goes on to explore the advantages which actually do result from a synthesis of science and art.

> 古典的な意味での「芸術」という言葉はこれだけです。実際、本のタイトルを選んだとき、私はこの意味で主に芸術を考えていませんでした、私は現在の意味合いをもっと考えていました。おそらく、否定的な検索で見つかった最も興味深い本は、Robert E. MuellerによるThe Science of Artと呼ばれるかなり最近の作品でした。 私が言及したすべての本の中で、ミューラーの本は、今日の私のテーマの中心テーマにしたいことを表現することに最も近い。彼は、「かつては芸術家の想像力に富んだ見方は科学者にとっては死であると考えられていた。科学の論理は「空想のすべての可能な芸術的飛行への破滅」を綴ったようだ」彼は、実際に科学と芸術の統合から生じる利点を探り続けています。

A scientific approach is generally characterized by the words logical, systematic, impersonal, calm, rational, while an artistic approach is characterized by the words aesthetic, creative, humanitarian,

anxious, irrational. It seems to me that both of these apparently contradictory approaches have great value with respect to computer programming.

> 一般的に、科学的アプローチは論理的、体系的、非人格的、冷静、合理的という言葉で特徴付けられますが、芸術的アプローチは美的、創造的、人道的、不安、不合理という言葉で特徴付けられます。これらの明らかに矛盾するアプローチはどちらも、コンピュータープログラミングに関して大きな価値があるように思えます。

Emma Lehmer wrote in 1956 that she had found coding to be "an exacting science as well as an intriguing art". H.S.M. Coxetcr remarked ira 1957 that he sometimes felt "more like an artist than a scientist". This was at the time C.P. Snow was beginning to voice his alarm at the growing polarization between "two cultures" of educated people. He pointed out that we need to combine scientific and artistic values if' we are to make real progress.

> エマ・レーマーは1956年に、コーディングが「厳密な科学であると同時に興味をそそる芸術」であると発見したと書いています。H.S.M. Coxetcrはira 1957で、「科学者というより芸術家のようだ」と感じることがあると述べました。これは当時のC.P.雪は、教育を受けた人々の「二つの文化」の間の二極化の高まりに警鐘を鳴らし始めました。彼は、私たちが真の進歩を遂げるには、科学的価値と芸術的価値を組み合わせる必要があると指摘しました。

Works of Art

When I'm sitting in an audience listening to a long lecture, my attention usually starts to wane at about this point in the hour. So I wonder, are you getting a little tired of my harangue about "science" and "art"? I really hope that you'll be able to listen carefully to the rest of this, anyway, because now comes the part about which I fled most deeply.

> 長い講義を聴いている聴衆に座っているとき、私の注意は通常1時間のこの時点で薄れ始めます。「科学」と「芸術」についての私の言い回しに少しうんざりしていますか？とにかく、これから私が最も深く逃げた部分が来るので、あなたがこれの残りを注意深く聞くことができることを本当に望みます。

When I speak about computer programming as an art, I am thinking primarily of it as an art form, in an aesthetic sense. The chief" goal of my work as educator and author is to help people learn how to write beautiful programs. It is for this reason I was especially pleased to learn recently that my books actually appear in the Fine Arts Library at Cornell University. (However, the three volumes apparently sit there neatly on the shelf, without being used, so I'm afraid the librarians may have made a mistake by nterpreting my title literal!y.)

> アートとしてのコンピュータープログラミングについて話すとき、私はそれを美的な意味でのアート形式として主に考えています。 教育者および著者としての私の仕事の主な目標は、人々が美しいプログラムを書く方法を学ぶのを助けることです。このため、最近、私の本が実際にコーネル大学の美術図書館に掲載されていることを知って、とても嬉しかったです。（ただし、3つのボリュームは使用されることなく棚の上にきちんと置かれているようです。そのため、図書館員がタイトルリテラルを間違えて間違えたのではないかと心配しています。）

My feeling is that when we prepare a program, it can be like composing poetry or music; as Andrei Ershov has said, programming can give us both intellectual and emotional satisfaction, because it is a real achievement to master complexity and to establish a system of consistent rules.

> 私の感覚では、プログラムを準備するとき、詩や音楽を作曲するようなものになります。アンドレイ・エルショフが言ったように、プログラミングは、複雑さを克服し、一貫したルールのシステムを確立することは本当の成果であるため、知的満足と感情的満足の両方を与えることができます。

Furthermore when we read other people's programs, we can recognize some of them as genuine works of art. I can still remember the great thrill it was for me to read the listing of Stan Poley's SOAP II assembly program in 1958; you probably think I'm crazy, and styles have certainly changed grcaltiy since then, but at the time it meant a great deal to me to see how elegant a system program could be, especially by comparison with the heavy-handed coding found in other listings I had been studying at the same time. The possibility of writing beautiful programs, even in assembly language, is what got me hooked on programming in the first place.

> さらに、他の人のプログラムを読むとき、それらのいくつかを本物の芸術作品として認識することができます。1958年にスタンポーリーのSOAP IIアセンブリプログラムのリストを読んだことは、私にとって大きなスリルを今でも覚えています。おそらく私はおかしいと思うし、それ以来スタイルは確かにgrcaltiyに変わったが、その時は、特に他の人に見られるヘビーハンドのコーディングと比較して、システムプログラムがどれほどエレガントであるかを見ることができた私が同時に勉強していたリスト。アセンブリ言語でさえ美しいプログラムを書く可能性は、そもそもプログラミングに夢中になったものです。

Some programs are elegant, some are exquisite, some are sparkling. My claim is that it is possible to write grand programs, treble programs, truly magnificent ones!

> エレガントなプログラムもあれば、絶妙なプログラムもあり、輝くプログラムもあります。 私の主張は、壮大なプログラム、高音のプログラム、本当に素晴らしいものを書くことができるということです！

Taste and Style

The idea of style in programming is now coming to the forefront at last, and I hope that most of you have seen the excellent little book on Elements of Programming Style by Kernighan and Plauger. In this connection it is most important for us all to remember that there is no one "best" style; everybody has his own preferences, and it is a mistake to try to force people into an unnatural mold. We often hear' the saying, "I don't know anything about art, but I know what I like." The important thing is that you really like the style you are using; it should be the best way you prefer to express yourself.

> プログラミングのスタイルのアイデアがついに最前線に来ており、KernighanとPlaugerによるElements of Programming Styleについての素晴らしい小さな本を見たことを願っています。 これに関連して、誰もが「最良の」スタイルは存在しないことを覚えておくことが最も重要です。誰もが自分の好みを持っているので、人々を不自然な型に強制しようとするのは間違いです。 「芸術については何も知りませんが、好きなことは知っています。」という格言をよく耳にします。重要なことは、使用しているスタイルが本当に好きなことです。それはあなたがあなた自身を表現することを好む最良の方法であるべきです。

Edsger Dijkstra stressed this point in the pretence to his Short Introduction to the Art of Programming:

> Edsger Dijkstraは、この点を強調して、彼の「プログラミングアート入門」への忍耐力を強調しました。

It is my purpose to transmit the irnportance of good taste and style in programming, but the specific elements of style presented serve only to illustrate what benefits can be derived from "style" in general. In this respect I feel akin to the teacher of composition at a conservatory: He does not teach Ms pupils how to compose a particular symphony, he must help his pupils to find their own style and must explain to them what is implied by this. (It has been this analogy that made me talk about "The Art of Programming.")

> プログラミングの趣味とスタイルの重要性を伝えることは私の目的ですが、提示されたスタイルの特定の要素は、一般に「スタイル」からどのような利点が得られるかを示すだけです。 この点で、私は音楽院の作曲の先生に似ていると感じています：彼は生徒に特定の交響曲を作曲する方法を教えていません。（この類推により、「プログラミングの技術」について話すようになりました。）

Now we must ask ourselves, What is good style, and what is bad style? We should not be too rigid about this in judging other people's work. The early nineteenth-century philosopher Jeremy Bentham put it this way:

> 今、私たちは自分自身に尋ねなければなりません、良いスタイルとは何ですか、そして悪いスタイルは何ですか？私たちは他の人の作品を判断する際にこれについて厳しすぎるべきではありません。19世紀初頭の哲学者ジェレミーベンサムは次のように述べています。

Judges of elegance and taste consider themselves as benefactors to the human race, whilst they are really only the interrupters of their pleasure. There is no taste which deserves the epithet good, unless it be the taste for such employments which, to the pleasure actually produced by them, conjoin some contingent or future utility: there is no taste which deserves to be characterized as bad, unless it be a taste for some occupation which has a mischievous tendency.

> 優雅さと嗜好の裁判官は、自分自身を人類の恩人と見なしますが、彼らは本当に彼らの喜びの妨害者にすぎません。それが実際に彼らが生み出した喜びに、偶発的または将来の有用性を結びつけるような雇用のための味でなければ、それはそうでない限り、悪いと特徴付けられるに値する味はありませんいたずら傾向のある職業の好み。

When we apply our own prejudices to "reform" someone else's taste, we may be unconsciously denying him some entirely legitimate pleasure. That's why I don't condenm a lot of things programmers do, even though I would never enjoy doing them myself. The important thing is that they are creating something they feel is beautiful.

> 他人の好みを「改革」するために自分自身の偏見を適用するとき、私たちは無意識のうちに彼に完全に正当な喜びを否定するかもしれません。だからこそ、プログラマーが自分でやることを決して楽しんでいないとしても、プログラマーがしていることの多くを納得させないのです。重要なことは、彼らが美しいと感じるものを作成しているということです。

In the passage I just quoted, Bentham does give us some advice about certain principles of aesthetics which are better than others, namely the "utility" of the result. We have some freedom in setting up our personal standards of beauty, but it is especially nice when the things we regard as beautiful are also regarded by other people as useful. I must confess that I really enjoy writing computer programs; and I especially enjoy writing programs which do the greatest good, in some sense.

> 先ほど引用した箇所で、ベンサムは私たちに、他のものよりも優れた美学の特定の原則、すなわち結果の「有用性」についていくつかのアドバイスを与えてくれました。私たちは個人的な美しさの基準を設定する自由がありますが、私たちが美しいとみなすものが他の人からも有用であるとみなされるとき、それは特に素晴らしいです。私はコンピュータープログラムを書くのが本当に楽しいと告白しなければなりません。そして、ある意味で最大の利益をもたらすプログラムの作成を特に楽しんでいます。

There are many senses in which a program can be "good," of course. In the first place, it's especially good to have a program that works correctly. Secondly it is oken good to have a program that won't be hard to change, when the time for adaptation arises. Both of these goals are achieved when the program is easily readable and understandable to a person who knows the appropriate language.

> もちろん、プログラムが「良い」ことには多くの感覚があります。そもそも、正しく動作するプログラムを用意しておくと特に便利です。第二に、適応の時が来たときに、変更するのが難しくないプログラムを持つことは良いことです。これらの両方の目標は、適切な言語を知っている人がプログラムを簡単に読み取り、理解できる場合に達成されます。

Another important way for a production program to be good is for it to interact gracefully with its users, especially when recovering fl'om human errors in the input data. lt's a real art to compose meaningful error messages or to design flexible input formats which are not error-prone.

> 本番プログラムが適切であるための別の重要な方法は、特にユーザーのエラーを回復するときに、ユーザーと上品にやり取りすることです。入力データ。意味のあるエラーメッセージを作成したり、エラーが発生しにくい柔軟な入力形式を設計したりするのは本物です。

Another important aspect of program quality is the efficiency with which the computer's resources are actually being used. I am sorry to say that many people nowadays are condemning program efficiency, telling us that it is in bad taste. The reason for this is that we are now experiencing a reaction from the time when efficiency was the only reputable criterion of goodness, and programmers in the past have tended to be so preoccupied with efficiency that they have produced needlessly complicated code; the result of this unnecessary complexity has been that net efficiency has gone down, due to difficulties of debugging and maintenance.

> プログラムの品質のもう1つの重要な側面は、コンピューターのリソースが実際に使用される効率です。最近、多くの人がプログラムの効率性を非難していることを残念に思っています。その理由は、効率が唯一の評判の良い基準であった時代からの反応を経験しているためであり、過去のプログラマーは、不必要に複雑なコードを生成するほど効率に没頭する傾向がありました。この不必要な複雑さの結果は、純効率がなくなったことです。デバッグとメンテナンスが困難なため、ダウンしています。

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

> 本当の問題は、プログラマーが間違った場所で間違った時間に効率を心配しすぎているということです。早すぎる最適化は、プログラミングにおけるすべての悪（または少なくともその大部分）の根源です。

We shouldn't be penny wise and pound foolish, nor should we always think of efficiency in terms of so many percent gained or lost in total running time or space. When we buy a car, many of us are almost oblivious to a difference of $50 or $100 in its price, while we might make a special trip to a particular store in order to buy a 50¢ item for only 25¢. My point is that there is a time and place for efficiency; I have discussed its proper role in my paper on structured programming, which appears in the current issue of Computing Surveys.

> 私たちは一文惜しみの百失いになってはなりません。また、実行時間やスペース全体で得られる、または失われる割合が非常に多いという点で、効率を常に考えるべきではありません。 車を買うとき、私たちの多くはその価格の差額50ドルまたは100ドルにほとんど気づいていません。一方、たった25¢で50¢のアイテムを買うために特定の店に特別な旅行をするかもしれません。私のポイントは、効率のための時間と場所があるということです。 Computing Surveysの最新号に掲載されている構造化プログラミングに関する論文で、その適切な役割について説明しました。

Less Facilities: Mere Enjoyment

One rather ,curious thing I've noticed about aesthetic satisfaction is that our pleasure is significantly enhanced when we, accomplish something with limited tools. For example, the program of which I personally am most pleased and proud is a compiler I once wrote for a primitive minicomputer which had only 4096 words of memory, 16 bits per word. It makes a person feel like a real virtuoso to achieve something under such severe restrictions.

> 審美的な満足度について気づいたかなり奇妙なことの1つは、限られたツールで何かを達成するとき、私たちの喜びが大幅に向上することです。 たとえば、私が個人的に最も喜んで誇りに思っているプログラムは、かつて4096語しかなかった原始的なミニコンピューター用に書いたコンパイラーです。 メモリ、16ビット/ワード。そのような厳しい制限の下で何かを達成することは、人を本当の名手のように感じさせます。

A similar phenomenon occurs in many other contexts. For example, peopIe often seem to fall in love with their Volkswagens but rarely with their Lincoln Continentals (which presumably run much better). When I learned programming, it was a popular pasthne to do as much as possible with programs that fit on only a single punched card. I suppose it's this same phenomenon that makes APL enthusiasts relish their "one-liners." When we teach programming nowadays, it is a curious fiact tlhat we rarely capture the heart of a student for computer science until he has taken a course which allows "hands on" experience with a minicomputer. The use of our large-scale machines with their fancy operating systems and languages doesn't really seem to engender any love for programming, at least not at first.

> 同様の現象は、他の多くの状況でも発生します。 たとえば、人々はフォルクスワーゲンと恋に落ちることが多いようですが、リンカーンコンチネンタル（めったに走らないと思われる）とめったに恋をしません。 プログラミングを学んだとき、パンチされた1枚のカードだけに収まるプログラムで可能な限り多くのことをするのが一般的でした。 APLマニアが彼らの「ワンライナー」を楽しんでいるのは、これと同じ現象だと思います。 私たちが最近プログラミングを教えるとき、彼がコースを受講するまで、コンピュータサイエンスの学生の心をつかむことはめったにありません。 これにより、ミニコンピューターでの「実践」体験が可能になります。 派手なオペレーティングシステムと言語を備えた大規模なマシンの使用は、少なくとも最初はプログラミングへの愛情を生むようには見えません。

It's not obvious how to apply this principle to increase programmers' enjoyment of their work. Surely programmers would groan if their manager suddenly announced that the new machine wilt

have only half as much memory as the old. And I don't think anybody, even the most dedicated "programming artists," can be expected to welcome such a prospect, since nobody likes to lose facilities unnecessarily. Another example may help to clarify the situation: Film-makers strongly resisted the introduction of talking pictures in the 1920's because they were justly proud of the way they could convey words without sound. Similarly, a true programming artist might well resent the introduction of more powerful equipment; today's mass storage devices tend to spoil much of the beauty of our old tape sorting methods. But today's film makers don't want to go back to silent films, not because they're lazy but because they know it is quite possible to make beautiful movies using the improved technology. The form of their art has changed, but there is still plenty of room for artistry.

> この原則をプログラマーの仕事の楽しみを増やすためにどのように適用するかは明らかではありません。新しいマシンのメモリが古いメモリの半分にしかならないことをマネージャーが突然発表すると、プログラマはうめき声を上げるでしょう。そして、誰でも、最も熱心な「プログラミングアーティスト」でさえも、誰も不必要に施設を失うことを好まないので、そのような見通しを歓迎すると予想されます。別の例は状況を明確にするのに役立つかもしれません：映画製作者は、音なしで言葉を伝えることができる方法を誇りに思っていたため、1920年代に写真を話すことの導入に強く抵抗しました。同様に、真のプログラミングアーティストは、もっと多くの 強力な機器;今日の大容量記憶装置は、古いテープソート方法の美しさの多くを損なう傾向があります。しかし、今日の映画製作者は、彼らが怠けているからではなく、改良された技術を使用して美しい映画を作ることがかなり可能であることを知っているため、サイレント映画に戻りたくありません。彼らの芸術の形は変わりましたが、芸術性の余地はまだたくさんあります。

How did they develop their skill? The best film makers through the years usually seem to have learned their art in comparatively primitive circumstances, often in other countries with a limited movie industry. And in recent years the most important things we have been learning about programming seem to have originated with people who did not have access to very large computers. The moral of this story, it seems to me, is "that we should make use of the idea of" limited resources in our own education. We can all benefit by doing occasional "toy" programs, when artificial restrictions are set up, so that we are forced to push our abilities to the limit. We shouldn't live in the lap of luxury all the time, since that tends to make us lethargic. The art of tackling miniproblems with all our energy will sharpen our talents for the real problems, and the experience will help us to get more pleasure From our accomplishments on less restricted equipment.

> 彼らはどのようにスキルを伸ばしましたか？長年にわたって最高の映画製作者は、通常、比較的原始的な状況で、多くの場合映画産業が限られている他の国で自分の芸術を学んだようです。 そして近年、私たちがプログラミングについて学んできた最も重要なことは、非常に大きなコンピューターにアクセスできなかった人々に由来しているようです。この話の教訓は、私たち自身の教育で「限られた資源を利用するべきだ」ということです。人為的な制限が設定されているとき、私たちは時折「おもちゃ」プログラムを行うことですべての利益を得ることができます。私たちは無気力になる傾向があるので、私たちは常に贅沢のラップに住むべきではありません。すべてのエネルギーでミニ問題に取り組む技術は、実際の問題に対する才能を磨き、経験は、制限の少ない機器での成果からより多くの喜びを得るのに役立ちます。

In a similar vein, we shouldn't shy away From "art for art's sake"; we shouldn't fed guilty about programs that are just for fun. I once got a great kick out of" writing a one-statement ALGOL program that invoked an innerproduct procedure in such an unusual way that it calculated the ruth

prime number, instead of an innerproduct. Some years ago the students at Stanford were excited about finding the shortest FORTRAN program which prints itself out, in the sense that the program's output is identical to its own source text. The same problem was considered for many other languages. I don't think it was a waste of time for them to work on this; nor would Jeremy Bentham, whom I quoted earlier, deny the "utility" of such pastimes. "On the contrary," he wrote, "there is nothing, the utility of which is more incontestable. To what shall the character of utility be ascribed, if not to that which is a source of pleasure?"

> 同様に、「芸術のための芸術」から遠ざかるべきではありません。私たちはただの楽しみのためだけのプログラムについて罪を犯すべきではありません。私はかつて、内積ではなくルース素数を計算するような異常な方法で内積手続きを呼び出した1ステートメントのALGOLプログラムを書くことから素晴らしいキックを得ました。数年前、スタンフォード大学の学生は、プログラムの出力が独自のソーステキストと同一であるという意味で、それ自体を印刷する最短のFORTRANプログラムを見つけることに興奮していました。同じ問題が他の多くの言語で考慮されました。 彼らがこれに取り組むのは時間の無駄ではないと思います。先ほど引用したジェレミーベンサムも、そのような娯楽の「実用性」を否定しません。「それどころか、彼は書いた、「その効用はより議論の余地のないものはない。快楽の源ではないにしても、効用の性格は何に帰すべきか？」

Providing Beautiful Tools

Another characteristic of modern art is its emphasis on creativity. It seems that many artists these days couldn't care less about creating beautiful things; only the novelty of an idea is important. I'm not recommending that computer programming should be like modern art in this sense, but it does lead me to an observation that I think is important. Sometimes we are assigned to a programming task which is almost hopelessly dull, giving us no outlet whatsoever for any creativity; and at such times a person might well come to me and say, "So programming is beautiful? It's all very well fbr you to declaim that I should take pleasure in creating eIegant and charming programs, but how am i supposed to make this mess into a work of art?"

> 現代美術のもう一つの特徴は、創造性を重視していることです。最近の多くのアーティストは、美しいものを作ることにそれほど関心を払っていなかったようです。重要なのはアイデアの新規性のみです。この意味で、コンピュータープログラミングを現代美術のようなものにすることはお勧めしませんが、それが重要だと思うことに気付きました。時々、私たちはほとんど絶望的に退屈なプログラミングタスクに割り当てられ、創造性のための出口をまったく与えません。そして、そのような時に、人が私のところに来て、「だからプログラミングは美しいのか？」と言うかもしれません。私は、とても魅力的で魅力的なプログラムを作ることに喜びを感じるべきだと断言するのはとても良いことですが芸術作品？」

Welt, it's true, not all programming tusks are going to be fun. Consider the "trapped housewife," who has to clean off the same table every day: there's not room for creativity or artistry in every situation. But even in such cases, there is a way to make a big improvement: it is still a pleasure to do routine jobs if we have beautiful things to work with. For example, a person will really enjoy wiping off" the dining room table, day after day, if it is a beautifully desig~lcd ruble made from some fine quality hardwood.

> 確かに、すべてのプログラミングタスクが楽しくなるわけではありません。毎日同じテーブルを片付けなければならない「閉じ込められた主婦」を考えてみましょう。あらゆる状況に創造性や芸術性の余地はありません。しかし、そのような場合でさえ、大きな改善を行う方法があります。美しいもの

> があれば、いつもの仕事をするのはまだ楽しみです。例えば、上質の広葉樹から作られた美しくデザインされたルーブルの場合、人は食堂のテーブルを毎日拭いて本当に楽しんでいます。

Therefore I want to address my closing remarks to the system programmers and the machine designers who produce the systems that the rest of us must work with. Please, give us tools that are a pleasure to use, especially for out routine assignments, instead of providing something we have to fight with. Please, give us tools that encourage us to write better programs, by enhancing our pleasure when we do so.

> したがって、私は、システムプログラマーと、他の人たちと一緒に作業しなければならないシステムを生産する機械設計者に、私の最後の挨拶を述べたいと思います。私たちと戦わなければならないものを提供する代わりに、特に日常的な課題のために使用する喜びのツールを教えてください。私たちがそうするときに喜びを高めることによって、より良いプログラムを書くことを奨励するツールを私たちに与えてください。

It's very hard for me to convince college fleshmen that programming is beautiful, when the first thing I have to tell them is how to punch "slash slash JOB equals so-and-so." Even job control languages can be designed so that they are a pleasure to use, instead of being strictly functional.

> 大学の人にプログラミングが美しいと納得させるのは非常に難しいです。最初に言わなければならないのは、「スラッシュスラッシュJOBはまあまあです」ということです。ジョブ制御言語でさえ、厳密に機能するのではなく、使いやすいように設計できます。

Computer hardware designers can make their machines much more pleasant to use, for example by providing floating-point arithmetic which satisfies simple mathematical laws. The facilities presently available on most machines make the job of rigorous error analysis hopelessly difficult, but properly designed operations would encourage numerical analysts to provide better subroutines which have certified accuracy.

> コンピューターハードウェアの設計者は、たとえば単純な数学の法則を満たす浮動小数点演算を提供することにより、マシンをより使いやすくすることができます。現在、ほとんどのマシンで利用可能な機能は、厳密なエラー分析の仕事を絶望的に困難にしますが、適切に設計された操作は、数値分析者が精度を証明したより良いサブルーチンを提供することを奨励します。

Let's consider also what software designers can do. One of the best ways to keep up the spirits of a system user is to provide routines that he can interact with. We shouldn't make systems too automatic, so that the action always goes on behind the scenes; we ought to give the programmer-user a chance to direct his creativity into useful channels. One thing all programrners have in common is that they enjoy working with machines; so let's keep them ira the loop. Some tasks are best done by machine, while others are best done by human insight; and a properly designed system will find the right balance. (I have been trying to avoid misdirected automation for many years.)

> ソフトウェア設計者ができることも考えてみましょう。システムユーザーの精神を維持するための最良の方法の1つは、ユーザーが対話できるルーチンを提供することです。システムがあまりにも自動化されないようにして、アクションが常に舞台裏で行われるようにしてください。プログラマーユーザーに彼の創造性を有用なチャンネルに向ける機会を与えるべきです。すべてのプログラマーに共通していることの1つは、マシンでの作業を楽しんでいるということです。ループを維持します。一部のタスクは機械によって最もよく実行されますが、他のタスクは人間の洞察によって最もよく実行さ

> れます。適切に設計されたシステムが適切なバランスを見つけます。（私は長年、誤った方向の自動化を避けようとしてきました。）

Program measurement tools make a good case in point. For years, programmers have been unaware of how the real costs of computing are distributeci in their prograrns. Experience indicates that nearly everybody has the wrong idea about the real bottlenecks in his programs; it is no wonder that attempts at efficiency go awry so often, when a programmer is never given a breakdown of costs according to the lines of code he has written. His job is something Eke that of a newly married couple who try to plan a balanced budget without knowing how much the individual items like food, shelter, and clothing will cost. All that we have been giving programmers is an optimizing compiler, which mysteriously does something to the programs it translates but which never explains what it does. Fortunately we are now finalIy seeing the appearance of systems which give the user credit for some intellb gence; they automatically provide instrumentation of programs and appropriate feedback about the real costs. These experimental systems have been a huge success, because they produce measurable improvemeats, and especially because they are fun to use, so I am confident that it is only a matter of time before the use of such systems is standard operating procedure. My paper in ComputitTg Surveys discusses this further, and presents some ideas for other ways in which an appropriate interactive routine can enhance the satisf'action of user programmers.

> プログラム測定ツールがその良い例です。何年もの間、プログラマは、コンピューティングの実際のコストがプログラムの中でどのように分配されているかを知りませんでした。経験から、ほぼすべての人が自分のプログラムの本当のボトルネックについて間違った考えを持っていることがわかります。プログラマーが書いたコードの行に応じたコストの内訳を決して与えられないとき、効率の試行がしばしば失敗するのは不思議ではありません。彼の仕事は、食料、避難所、衣類などの個々のアイテムの費用がわからないまま、バランスの取れた予算を計画しようとする新婚カップルの仕事です。プログラマーに提供しているのは、最適化コンパイラーだけです。最適化コンパイラーは、翻訳するプログラムに何か不思議なことをしますが、それが何をするのかを説明しません。幸いなことに、私たちは最終的に、ユーザーに何らかの知恵を与えるシステムの出現を見ています。プログラムの計測と実際のコストに関する適切なフィードバックを自動的に提供します。これらの実験システムは、測定可能な改善肉を生成するため、特に使用するのが楽しいため、大成功を収めています。そのようなシステムの使用が標準的な操作手順になるのは時間の問題だと確信しています。ComputitTg Surveysでの私の論文では、これについてさらに説明し、適切な対話型ルーチンがユーザープログラマーの満足度を高める他の方法についてのいくつかのアイデアを提示します。

Language designers also have an obligation to provide languages that encourage good style, since we all know that style is strongly influenced by the language in which it is expressed. The present sttrge of interest ira structured programming has revealed that none of our existing languages is really ideal for dealing with program and data structure, nor is it clear what an ideal language should be. Therefore I look forward to many careful experiments in language design during the next few years.

> 言語設計者には、スタイルが言語に強く影響されることは誰もが知っているため、良いスタイルを奨励する言語を提供する義務もありますそれが表現されています。関心のある現在の構造ira構造化プログラミングは、既存の言語のどれもプログラムやデータ構造を扱うのに本当に理想的ではなく、理想的な言語がどうあるべきか明確でもないことを明らかにしました。したがって、私は今後数年間、言語設計の多くの慎重な実験を楽しみにしています。

Summary

To summarize: We have seen that computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better. Therefore we can be glad that people who lecture at computer conferences speak about the state of the Art.

> 要約すると、コンピュータープログラミングは芸術であり、蓄積された知識を世界に適用するためです。創意工夫、そして特にそれが美しさのオブジェクトを生成するため。自分をアーティストとして無意識のうちに見ているプログラマーは、自分がしていることを楽しんで、もっとうまくやるでしょう。したがって、コンピューター会議で講演する人々が最新技術について語るのは嬉しいことです。