Draft of ECOOP ‘99 Banquet Speech, Peter Wegner

In 1967 I taught at Cornell, was involved in the development of Curriculum 68, and was working on my book, *Programming Languages, Information Structures, and Machine Organization*, which aimed to develop a unifying model for programming languages. When Don Knuth visited me one weekend in Ithaca, we both went to the synagogue on Saturday and to Church on Sunday and had intense discussions about the nature of computer science. Don was then working on the first volume of *The Art of Computer Programming* and was the leading exponent of the view that computer science was the study of algorithms. Don’s bottom-up approach to computer science, namely by constructing composite structures from primitives, contrasted strongly with my top-down approach of classifying and comparing already existing programming langauges. Don and I discussed the possibility of writing a joint book on data structures, but concluded that the gulf between bottom-up and top-down computer science was at this time too broad to be bridged.

I first heard about Simula in 1967 and immediately fell in love with object-oriented programming, including a section on the relation between Simula ‘67 and Algol 60 in my 1968 book. Object-oriented programming encourages a top-down view of the world in terms of persistent objects, in contrast to algorithms that model the world bottom-up by time-independent input-output transformations.

My early interest in object-oriented programming led to a close relationship with Kristen Nygaard who, like me, was a top-down researcher. At a panel discussion on the nature of inheritance at the Oslo ECOOP in 1988, Kristen asserted that inheritance models specification, while Danny Bobrow of the United States took the bottom-up position that inheritance models implementation. European computer scientists frequently take a top-down view that emphasizes principles and esthetics, while American computer scientists frequently take a bottom-up view based on practical and implementation considerations. My top-down approach is based partly on my European roots and European education.

My quest for top-down understanding led to a series of articles and edited books that explore research directions of computing. The 80-page 1970 article “Three Computer Cultures: Computer Mathematics, Computer Engineering, and Computer Science” considered computer science as a synthesis of the intellectual traditions of mathematics and engineering. My 1980s edited book “Research Directions in Software Engineering included a 80-page article “Research Directions in Programming Languages”. Here are some quotes from the introduction to this article:

“Programming languages simulataneously abstract from computers on which they are implemented and from applications which they are designed to model.

Abstraction from computers determines a model of implementation for type-indep[endent abstractions such as variables and assignment, while abstraction from applications determines type-depenmdent operations of specific data types. We may distinguish between the action-oriented view of programming languages exemplified by Algol 60 and the object-oriented view of Simula and Smalltalk."

In 1987 I edited "Research Directions in Object-Oriented programming" with an 80-page article "The Object-Oriented Classification Paradigm" that attributed the success of object-oriented programming to the seamless integration of state-transition, communication, and classification paradigms. My 1989 OOPSLA keynote talk resulted in yet another 80-page paper "Concepts and Paradigms of Object-Oriented Programming" that explored wide ranging sequential and concurrent language design issues as well as mathematical models for objects, classes, and inheritance.

This pattern of exploration of top-down questions in computing by long 80-page papers can be interpreted in a variety of ways, and I am never sure whether people who tell me I am a classifier or synthesizer are being complementary. Top-down research is riskier than bottom-up research and is not recommended for PhDs, but has the potential of discovering big ideas.

During the 1970s and 1980s, I felt that the gulf between top-down and bottom-up CS was due to the immaturity of the field and would in time be bridged by better semantic models and formal methods technology. My research on semantics in the 1970s and on object-oriented programming in the 1980s was motivated by bridging this gap. But in the 1990s I realized that the algorithm model was inherently too weak to express the behavior of object-oriented and distributed systems, so that this gap could not be bridged even in principle. Objects express persistent services over time which cannot be modeled by time-independent input-output actions.

This insight that Turing machines cannot express interactive computation leads to a radical paradigm-shift. It requires the reexamination of fundamental assumptions about the nature of computer science. For example, Church's thesis, which asserts that the intuitive notion of computing is that of Turing machines, must be revised. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, our research questions the validity of the broader assertion that algorithms capture the intuitive notion of what computers compute. Interactive models capture the notion of persistent computing agents that provide services over time which are not expressible as computable functions.

I first presented the idea that Turing machines cannot model interaction at the 1992 closing conference of the Japanese 5th generation computing project, showing that the project's failure to reducing computation to logic was due not to lack of cleverness on the part of logic programming researchers, but to theoretical impossibility of such a reduction. The key argument is the inherent tradeoff between logical completeness and commitment. Committed choice to a course of action is inherently incomplete because commitment cuts off branches of the proof tree that might contain the solution, and commitment is therefore incpmpatible with complete exploration. However, commitment is essential to object-oriented functionality, since execution of a message by an object cannot be reversed. Object-oriented flexibility can be achieved only by sacrificing logical completeness. Pure logic programming is inherently too weak to serve as a model for object-oriented programming.

The impossibility result that first-order logic is too weak to model object behavior is one instance of a family of impossibility results. This negative result has led to the development of positive models of interactive computation. Turing machines were extended to interaction machines and a metric of expressiveness based on observation power rather than transformation power was developed according to which interaction machines are more expressive than Turing machines.

Expressiveness for finite computing agents is defined in terms of the agent's ability to make observational distinctions about its environment. This notion of expressiveness applies equally to people and to computers. People who see are more expressive than blind people because they can make visual distinctions, while telescopes and microscopes increase expressiveness by allowing people to make still finer distinctions. The ability of interaction machines to make finer distinctions than Turing machines is illustrated by question answering. A questioner can learn more about a questioned person from a sequence of interactive questions than from a single question. For example, if Ken Starr is questioning Bill Clinton, Starr can learn more by interactive questioning with follow-up questions than by asking Clinton to fill in answers to a questionnaire, which has the status of a multi-part single question, since follow-up questions are not permitted.

The Turing test tests the ability of a noninteractive quesioner to distinguish Turing machine from human behavior. We extend the Turing test from the case where the machine is a Turing machine to the interactive Turing test, where the machine is an interaction machine. We show that interactive thinking as defined by the interactive Turing test is stronger than thinking definable by Turing's original test. The interactive Turing test allows questioned machines to adapt and learn, and to adapt answers to the needs of the questioner, just as a good lecturer adapts lecturing style to the needs of the audience. turing agents who can answer questions by interacting with the Internet or with chess experts through interfaces hidden from the questioner exhibit an even stronger form of thinking, which we call distributed thinking.

Critics like Searle and Penrose were right in arguing that the Turing test is too weak to model thinking when the object being questioned is a Turing machine, but wrong in asserting that thinking could not in principle be expressed by behavior. Turing's instincts were right in arguing that thinking can be modeled by behavior, but the behavioral model of thinking that he proposed was too weak.

Extending the Turing test to interaction not only helps us to understand the interactive nature of thought. It also lets us explore the relation between Turing machines and the Turing test, Turing's two big contributions. Turing machines and the Turing test are generally considered to be unrelated; but they are in fact closely related, in that Turing's definition of computing by a machine was the starting point of his definition of thought in terms of a precise notion of computing. Our extension of the notion of computation by an extended machine allows us to define a more powerful form of thinking by extending the Turing test.

Once we accept the idea that interaction is more powerful thna algorithms, we can distinguish multiple levels of expressiveness. In particular, interaction with multiple external agents or streams is more expressive than interaction with a single agent. This contrasts with the Turing machine result that multiple tapes are no more expressive than a sinlge tape. Interaction machiens with multiple streams are more expressive than a machine with a single stream, while Turing machines with multiple tapes are not more expressive than machiens with a single tape. This translates to the result that collaborative computing is more expressive than sequential inteer-action. A ceo who coordinates the activities of multiple subordiantes is more expressive than a junior employee with a single boss and no subordinates.

In the context of the Turing test this translates to the notion that sequential thinking by interaction with a single quesioner is less powerful than distributed thinking by interacting with multiple autonomous agents. In the context of databases this translates to the notion that nonserializable behavior is more expressive than serializable behavior. In the context of physics, it translates to the result that the three body problem is more expressive than the two-body problem. The result that interaction of an agent with multiple streams is more powerful than interaction with a single stream has ubiquitous interdiscipolinary implications.

The paradigm shift from algorithms to interaction is intimately related to the evolution of pracitcal computing. The technology shift from procedure-oriented, algorithmic mainframe technology of the 1970s to object-oriented and distributed, interactive workstation technology of the 1970s is fundamentally a shift from algorithms to interaction. The contract between computing agents and clients has changed from a sales contract that transforms inputs to outputs to a marriage contract that provides continuing services over time. The folk wisdom that marriage contracts cannot be expressed as sales contracts translates to the formal result that interaction machines cannot be modeled by Turing machines.

The irreducibility of interaction to algorithms explains Tim Rentsch’s comment that “everyone is talking about object-oriented programming but no one knows what it is”. This is as true today as it was 20 years ago, because it is impossible to define object-oriented programming in terms of algorithms. If “knowing what it is” means reducing object-oriented programming to algorithms then the reduction is bound to fail. But if we enlarge the class of things that “count” as explanations of object-oriented programming to include interactive models, then we can succeed.

Fred Brooks’ belief that there is no silver bullet for system specification can be restated in terms of the impossibility of algorithmic specification of interactive systems. In fact, a proof of irreducibility of interactive specifications to algorithms can actually be interpreted as a proof of the nonexistence of silver bullets.

There is abundant evidence that interactive services over time cannot be expressed by algorithms, but this result is difficult to prove theoretically. My colleague Dina Goldin and I have pursued two principal approaches towards this end, by machine models and mathematical models. Persistent Turing machines are a minimal extension of Turing machines that express interactive behavior by multitape machines with a persistent worktape preserved between interactions. Non-well-founded sets are an extension of traditional sets that provide a mathematical model of streams and interactive computing in terms of circular reasoning. The mathematics of circular reasoning was discovered only in the 1980s and is comprehensively presented for the fitrst time in the 1995 book Vicious Circles by Barwise and Moss. I will not try to explain non-well-founded set theory in a banquet speech, but will say a little about the intuitions that underlie circular reasoning and the reasons that it was not discovered and developed earlier.

The key intuition is that the class of things that a finite agent can observe is greater than the class of things that an agent can construct. We can formalize this intuition by showng that the class of things that an agent can construct is enumerable, while the set of situation that an agent can observe is nonenumerable. Moreover, we can show that constructible sets can be specified by induction, while observable sets require a stronger inference rule called coinduction.

Bertrand Russell in the 1900s and Hilbert and Godel in the 1920s made a fundamental mathematical mistake in assuming that induction was the strongest form of definition and reasoning. They were misled by the paradoxes of set theory and mitskenly thought that circular reasoning needed to describe observation processes was inconsistent. In fact, circular reasoning is consistent and allows stronger forms of definiton and reasoning than is possible through induction. Though induction is sufficient to describe construction processes stronger forms of reasoning are needed to express observation processes. Turing machines turn out to be the strongest form of computation possible by inductive reasoning but are not strong enough to express interactive computations of finite agents that observe an incompletely known environment, which are modeled by circular reasoning.

Inductive definitions of set theory and logic define minimal fixed points which exclude everything that is not explicitly definable, while coinductive definitions of non-well-founded set theory include everything that is not explicitly excluded. Observers who consider all possible worlds not ruled out by observations are using the coinductive maximal fixed point principle. The maximal class of things not explicitly excluded by a set of obsevations is fundamentally larger than the minimal class of things constructible from a set of primitives and allows us to build richer kinds of models.

Constructive models that employ induction can cope with only enumerable situations, while observation-based models that employ circular reasoning can cope with nonenumerable situations. Turing machines have only enumerable input strings and can perform only an enumerable number of computations, while interaction machines just like people can make nonenumerable distinctions about their environment. Interactive systems can handle nonenumerable environments while noninteractive systems can handle only enumerable environments.

The existence of a mathematical foundation for interactive computing provides a mathematical basis for interactive models of objects and distributed systems. Much work needs to be done on both the foundations and applications of interactive computing, but I believe the theory of computing in the 21st century will be very different from that in the 20th century amd that interactive models of finite agents will be grounded in mathematical theory. Textbooks will evolve that offer a balance between the study of algorithms and interaction, and provide a systematic foundation for interactive software technology.

Research on interaction is exciting because models of interaction provide a common foundation for software engineering, artificial intelligence, graphics, and HCI. It is exciting at an interdisciplinary level because it provides a precise definition and uniform approach to empirical computer science in terms of interactive models of computation and mathematics. It provides interesting perspectives on cognitive science through extensions of the Turing test and other interactive models. For example, it provides a broader perspective on intelligence testing by showing that traditional intelligence tests measuring algorithmic ability do not express interactive problem solving ability. Interactive intelligence, also known as emotional intelligence, has been shown to be a better predictor of success in later life than traditional intelligence testing. Interactive models provide a framework for explaining such results.

So I recommend to you to get involved in research on interactive models. They are a natural abstraction of object-oriented programming and provide an answer to Tim Rentsch's question about what OOP is. Interactive models are a wide open research area with many open practical and theoretical research problems. It is an incredibly exciting top-down research area with huge numbers of open bottom-up problems. I hope that this talk will encourage greater research on interactive models.