Bjarne Stroustrup

Bjarne Stroustrup designed and implemented C++, the first oo language widely adopted in industry. For more than a decade C++ has been the most widely used, industry-accepted, programming language supporting object-oriented programming. In the desgin an implementation of C++, Stroustrup addressed successfully the issues of designing a language that would be backwards compatible with C, would be type safe, and would allow for the emission of efficient code, through the design of simple algorithms.

His book "The C++ Programming Language" (Addison-Wesley, first edition 1985, second edition 1991, third edition 1997, "special" edition 2000) is most widely read book and has been translated into at least 19 languages. C++ has trongly influenced the design of newer oo languages, such as Java, C-sharp, and Fortran 99.

Stroustrup is working on the design of better C++ libraries, and of more power meachnisms for the design of abstract data types.

* Bjarne Stroustrup

Bjarne has earned a modicum of notoriety for being the designer and initial implementer of a programming language known as C++ (originally C with classes). This programming language, while arguably controversial, has had a significant impact on computing practices and has pushed object-oriented technology into the mainstream of computing. Bjarne has provided sustained leadership of the C++ standard throughout his career.

I would like to nominate the team behind Beta - with the exclusion of Kristen himself for obvious reasons.

I think that the team did a fantastic job of thinking thru the fundamentals of Object Oriented Programming and then designing a language incorporating a single, unified, fundamental concept for expressing an impressive number of the concepts of object oriented programming.

Not only did they design a new programming language, Beta, but they also provided a successful implementation of it that has been used in industry and for teaching at Aarhus University.

Albeit Kristen himself was a member of the team, the rest of the team had significant contribution especially in the actual implementation of the system.

Ole Lehman Madsen

Ole was the main implementor of the programming language Beta, a language in the Simula tradition. BETA is unique in that it supports classes, procedures, functions, coroutines, processes, and exception, under one abstraction mechanism: the pattern. In particular, he designed and implemented the mechanisms for type checking virtual patterns, wrote many books and articles on Beta, and brought the implementation to the maturity necessary for it to serve as the main as the main teaching language for "Programming in the Large" university courses, as well as industry courses. Finally, he has achieved a wide acceptance of Beta in the Danish industry.

Beta is widely used in Denmark, but less so outside Scandinavia. Nevertheless, Beta has created powerful concepts which have influenced oo language design throughout the world.

Martin Odersky

I'd like to nominate Martin Odersky, for his outstanding contribution to object-oriented programming through the design, implementation and evolution of the Scala language. The thread of publications of Martin since 2003, related to Scala, is just impressive. Every corner of Scala has been the subject of a top publication: the object model, composition, pattern matching, collections, continuations, type system, implicits, capabilities, concurrency, effects, embedded DSLs, compilation, reactivity. All this blended into a practical language that has gained and is still gaining a lot of traction in industry and in open source communities. Fundamentally, Scala is breaking new grounds in the integration of functional and object-oriented programming in a strongly-typed setting, largely contributing to educating programmers about the breadth and depth of advanced topics in programming.

Eric Tanter

Martin Odersky is a professor at EPFL in Lausanne, Switzerland, and an ACM fellow. He has described his special software passion as the unification of functions and objects, which is an area where he has contributed impressively over the years. The topic emerged in his papers already in 1991, two years after he received his PhD degree from ETH Zürich. Later on he was a main force behind the introduction of generics into the Java programming language, especially via his work on the languages Pizza and GJ. In fact, his GJ compiler was of such high quality that it was adopted by Sun Microsystems, Inc., as the standard javac compiler. More recently, the work of Martin Odersky and his research group has been organized around the creation and development of the Scala programming language. This language unifies the object-oriented and functional paradigm, and it supports many innovative features in type system concepts and mechanisms, as well as dynamic semantics and even syntax. The Scala language is also highly practical, not the least because of its smooth interoperability with Java. In general, Martin Odersky represents a rare combination of a deep understanding of foundational, theoretical issues, an effective and elegant approach to concrete programming language design, and a very practical sense of top quality software development, well documented in numerous research papers.

Main Contribution: The Scala language. This language embeds numerous novel language design and type system innovations, and at the same time the implementation of Scala is sufficiently robust to allow several companies to bet their business on its practical usefulness. Many papers have been written on research based on Scala, and several books describe the language for a broader audience.

(Previously indicated as main contribution: 'Making the future safe for the past: Adding genericity to the Java programming language' which has 346 citations according to http://academic.research.microsoft.com/Detail.aspx?entitytype=2&searchtype=2&id=1477294&orderBy=1).

I want to propose Alexander Summers, for his work on types, object invariants, object initialisation, the unification of implicit dynamic frames and separation logic, and

Alex obtained his PhD for work on the Curry Howard Isomorphism for Classical Logic in 2009. He continued this work after this PhD, but concentrated mainly on issues around oo languages, and the verification of oo programs.

He has worked on the use of Universe Types for the Verification of OO programs, and to do with the meaning of universe and ownership types and their use for program verification. He distilled elegant and adaptable models as well as approaches to their use for static variables. He has collaborated on the development of an elegant framework which explains seven different schemas which are based on various type systems and support different protocols for object invariants.

At the time however, there was grave scepticism about the role of object invariants in program verification, and in fact it was proposed that they were no more than a shorthand, and played no direct role in verification. Alex had the tenacity to investigate this question further. As a result, he wrote a study on the use of invariants based on a series of case studies, and argued their usefulness from the point of view of elegance and accessibility of the proofs. After that, based on this investigation, he proposed a novel verification pattern, Considerable Reasoning, which allows invariants to be broken in a more flexible manner than in universe types so far. Interestingly, Matthew Parkinson, who in 2007 had proposed that the concept of object invariant had become obsolete, started having doubt after reading these papers by Alex.

His work on object initialization and avoidance of null-pointer exception together with Peter Müller solves a ten year old problem, which had been tackled by many researchers already: how to statically check for the absence of null-pointer exceptions in a sound way without overburdening the programmer. In the area of type systems, and at a more fundamental level, he published work on models of existential types in collaboration with Mariangiola Dezani from Turin, and Nick Cameron from Mozilla.

Models for Program Verification
The work on the Relationship between Implicit Dynamic and Separation Logic with Matthew Parkinson solved a fundamental the question: how to relate the two most pervasive trends in program verification methodology at the time. He continued this work further in work on the semantics of recursive predicates.

Foundations and Tools for Program Verification
The work on Abstract Read Permissions, together with Stefan Heule, Peter Müller, and Rustan Leino from MSR, tackled an important practical problem: how to support multiple read permissions without overburdening the user with trivial permission calculation ? the fundamentals are developed and integrated into the Chalice tool. The work on Verification Condition Generation with Stefan Heule, Ioannis Cassios, and Peter Müller tackles the problem sound of verification condition generation for abstract predicates a functions ? a problem which had been attempted - and wrongly solved several times. Further work tackles the encoding of predicates across tools, and a practical solution to tackling the magic wand, a powerful, but computationally elusive logical connective.

More at http://people.inf.ethz.ch/summersa/wiki/doku.php?id=research

With this mail, I would like to nominate Jan Smans for the junior Dahl-Nygaard prize.

Jan obtained his PhD in 2009 on the topic of specification and verification of Java-like programs. He made several influential contributions to the field:

- he is the inventor of the concept of "implicit dynamic frames". The ECOOP paper on that concept [1] is the second most cited paper from the last 5 editions of ECOOP (source: Google Scholar metrics, at URL: http://scholar.google.be/citations?hl=en&vq=eng_softwaresystems&view_op=list_hcore&venue=-PVgS-XJp28J.2014
)

- he made substantial contributions to a number of formal verification systems, including VeriFast, Chalice, and VeriCool. With the VeriFast tool, Jan Smans (together with Bart Jacobs) was the winner of the Formal Methods 2012 verification competition.

I would be happy to answer any further questions you might have about this nomination.

Best regards,

Frank Piessens

[1] Jan Smans, Bart Jacobs, Frank Piessens, Implicit dynamic frames: Combining dynamic frames and separation logic, ECOOP 2009

I would like to nominate Arjun Guha for the _junior_ prize.

Arjun is one of the key people responsible for putting scripting languages on a sound footing. This is of high relevance to AITO because scripting languages have become the new carrier of object-orientation, in many ways hewing closer to some traditional notions of objects than even languages like Jav doa. However, they also have peculiar ways of capturing objects (usually as hash-tables) which leads to numerous differences, big and small, with traditional systems from Smalltalk to Java to Self.

Arjun's many contributions to scripting/OO include:

- the \JS semantics for JavaScript

This work, in ECOOP 2010, established a benchmark in multiple ways. First, it showed that the true complexity of languages like JavaScript is significantly greater than previous work had assumed. He also showed that it was feasible to nevertheless reduce this complex language to a simple core through compositional desugaring. Finally, he established the idea of a tested semantics by measuring the result against real language implementation test suites.

Beyond a paper result, \JS is an executable tool suite. As a result, it is now used in numerous projects, and several people outside the original research group continue to contribute to it. It has also spawned several variants. Even competing researchers have now fully embraced the idea of at ested semantics.

- static analysis for JavaScript

Arjun built one of the first static analyses for the full JavaScript language, and applied it to building control-flow intergrity monitors for server-side security.

- dataflow programming in JavaScript

Arjun was one of the co-leaders of the Flapjax project, which showed how to embed dynamic dataflow in JavaScript. This language has since influenced numerous other projects, including one from the Viewpoints Lab as well as two commerical systems.

- types for JavaScript

Arjun and his collaborators built a series of type systems for JavaScript, again embracing much of the language's full complexity. Indeed, because JavaScript cannot meaningfully have one single, canonical type system, eventually this resulted in the system TeJaS, a tool for building type systems for JavaScript.

In the process, Arjun also identified the importance of type refinement through both control-flow and state. He analyzed a corpus of code to show that this idea is common across several scripting languages, not only JavaScript.

Finally, Arjun's work demonstrated that traditional object type systems are insufficient for the needs of scripting languages. In contrast, he formulated a theory of objects with "first-class member names" -- i.e., objects whose member names could be computed values -- and presented both semantics and types for understanding this new style of objects.

- security analyses for Web sandboxes

Arjun applied all of the above ideas to perform the first verification of a real JavaScript security sandbox under genuine browser operating conditions. This not only found acknowledged security bugs in the system, there have been no further exploits since found that contradict the proof provided by this work.

As a result, Arjun's work has been influential in several areas: semantics, types, program analysis, and security. All these combined demonstrate his outsized influence in modern thinking about objects as represented by scripting languages. He is therefore a worthy contender for the junior prize.