Gespräch mit Bjarne Stroustrup

Tief verwurzelt in C++

Der Erfinder der Programmiersprache C++ Dr. Bjarne Stroustrup erzählt im Gespräch mit *iX Developer* von seinen Träumen für C++ und von der Bedeutung der Community für die Hochsprache, die seit vier Jahrzehnten wächst und gedeiht.

iX Developer: *C++ ist mittlerweile über 40 Jahre alt. Inmitten oft kurzlebiger Programmiersprachen ist das bemerkenswert. Warum ist es für Entwickler denn noch immer attraktiv, ausgerechnet diese Sprache zu lernen?*

Bjarne Stroustrup: Die Tatsache, dass C++ mehr als 40 Jahre alt ist und über eine aktive Community von rund 4,5 Millionen Entwicklern verfügt, ist schon für sich genommen ein starkes Indiz dafür, dass es sich lohnt, C++ zu lernen. Keine Sprache lebt doch so lange und zieht eine so große Gemeinschaft heran, ohne ein paar ernst zu nehmende Stärken aufzuweisen. Die meisten Programmiersprachen sterben jung oder bleiben in einem künstlichen Koma stecken. Unabhängig davon hat C++ Stärken, wenn es darum geht, eine Reihe nützlicher Konzepte und Techniken zu erlernen. Wenn du dein System auf Maschinenniveau kennen willst (Bits, Bytes, Wörter und Adresse), C++ ist gemacht dafür. Wenn du den Einsatz statischer Typen verstehen willst, C++ ist gemacht dafür. Wenn du dein Programm organisieren willst, indem du eigene Typen definierst und einsetzt: C++ ist gemacht dafür. Will man allgemeine Ressourcen verwalten wie Speicher, Dateien und Locks oder den eigenen Code für optimale Performance tunen, C++ ist gemacht dafür.

Dr. Bjarne Stroustrup ...

... ist der Vater von C++. Der Mathematiker und promovierte Informatiker ist Technical Fellow in der Technologieabteilung bei Morgan Stanley in New York City und Gastprofessor an der Columbia University. Er ist Mitglied der US-amerikanischen National Academy of Engineering (NAE) und Fellow weiterer technischer Gesellschaften. 2018 erhielt er den Charles-Stark-Draper-Preis der NAE für Ingenieurleistungen und 2017 die Faraday-Medaille. Als Forscher ist er den Bell Labs eng verbunden und seit über 30 Jahren wirkt er im ISO-Standard-Komitee mit. Sein Ziel ist es, C++ zu einer stabilen, zeitgemäßen Basis für Softwareentwicklung „für das echte Leben“ zu machen.

Ein anderer Punkt ist die Erkenntnis, dass C++ konventionelles Programmieren im Stil von C sowie Datenabstraktion, objektorientiertes, generisches und nebenläufiges Programmieren unterstützt. Noch ein Aspekt ist, dass C++ das Rückgrat großer Anwendungen bildet, und zwar in eingebetteten Systemen, Spielen, Finanzen, in der Biotechnologie, Medizin, Luft- und Raumfahrt sowie in der Automobilindustrie, bei künstlicher Intelligenz beziehungsweise Machine Learning und vielem anderen mehr. Zahlreiche der prominentesten Technologieunternehmen der Welt setzen C++ für ihre Kernsysteme ein.

Abgesehen davon sollten professionelle Entwickler mehr als nur eine Sprache beherrschen, und C++ – als eine davon – ist eine gute Wahl.

Konzentriert euch auf das Wesentliche

iX Developer: *Kannst du C++-Neulingen einen Rat geben, wie sie in diese anspruchsvolle Sprache einsteigen können?*

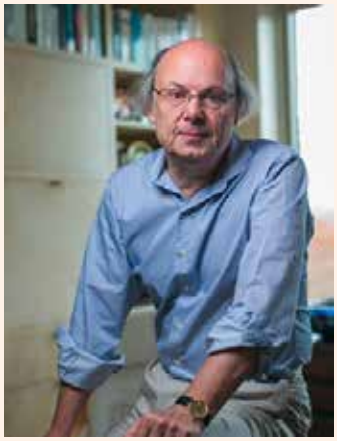
Stroustrup: Konzentriert euch auf das Wesentliche, auf die Konzepte, Prinzipien und Techniken. Verliert euch nicht in den unzähligen obskuren technischen Details. Versucht nicht, alles zu verstehen. Es ist schlicht unmöglich, alles auf einmal zu begreifen. Stellt euch das Beherrschen von C++, seinen Techniken, Bibliotheken und Werkzeugen wie das Erlernen einer neuen natürlichen Sprache vor, um darin ein ernsthaftes Gespräch zu führen, oder wie das Spielen eines Musikinstruments auf einem Niveau, dass jemand eurer Darbietung wirklich gern zuhören möchte. Um Abhilfe zu schaffen, habe ich zwei Bücher geschrieben:

- „Programming: Principles and Practice Using C++“ [1] für Menschen, die noch nie programmiert haben. Wie der Titel nahelegt, benutzt es C++ als Vehikel, um die Grundlagen des Schreibens guter Programme zu vermitteln.

- „A Tour of C++“ [2] ist für Menschen, die bereits durch eine andere Programmiersprache oder von einer älteren C++-Version gute Grundlagen im Programmieren haben. Es ist ein kurzer, 200 Seiten langer Überblick über die grundlegenden Möglichkeiten, die C++ bietet. Das Buch setzt voraus, dass du die Reife hast, selbstständig nach weiterem Material zu suchen, falls es nötig ist.

Beide Bücher liegen in zweiter Auflage vor und sind auch auf Deutsch und in einigen weiteren Sprachen erhältlich. Gute Programme zu schreiben benötigt eine gewisse Ausdauer sowie Verständnis des Bereichs der Anwendungen und der an ihnen beteiligten Toolchains. Oft ist die Programmiersprache nicht einmal das komplizierteste und kniffligste Thema, das es zu beherrschen gilt, um etwas Nützliches herzustellen.

Quelle: stroustrup.com



Bjarne Stroustrup in seinem Arbeitszimmer in New York (2018)

Gut gemacht ist das Programmieren – die Softwareentwicklung – eine hohe Kunst und zugleich eine sehr nützliche. Es lohnt sich, beträchtlichen Aufwand in ihre Beherrschung zu investieren. Stellt sie euch vor wie Mathematik oder Physik, nicht wie eine niederschwellige Fertigkeit, die man in ein paar Wochen erwirbt. C++ kann Kernstück solch einer Suche nach Meisterschaft sein.

Frauen in der IT und in der C++-Community

iX Developer: *C++ wird von vielen noch immer vorrangig als „Männerdomäne“ betrachtet. Wie siehst du das, Bjarne?*

Stroustrup: Das ist eine große Schande! Schlimmer noch, in manchen Bereichen ist das nicht nur eine Wahrnehmung, sondern wahr. Sollte es aber nicht sein. Noch bitterer, hier geht es nicht nur um C++, das ist in weiten Teilen der IT und Informatik der Fall. Wir müssen etwas dagegen tun, aber bislang hat sich das als überraschend schwierig herausgestellt.

iX Developer: *Wie würdest du mehr Frauen als Entwicklerinnen dazu ermutigen, in C++ einzutauchen?*

Stroustrup: Das ist offensichtlich leichter gesagt als getan, sonst hätten wir das Problem vor Jahrzehnten schon gelöst. Unzweifelhaft gibt es eine Menge Gründe für diesen erbärmlichen Zustand, aber von dort aus, wo ich sitze, scheint es zum großen Teil mehr ein Problem der Wahrnehmung zu sein als etwas Feindliches oder Übergriffiges vonseiten der Informatik und der IT-Leute. Zweifellos ist nicht jeder in der IT, Informatik und C++-Community perfekt, aber im Durchschnitt können wir doch nicht schlimmer sein als so viele andere Bereiche, wo Frauen mittlerweile in großer Zahl Erfolg haben, etwa in der Medizin.

Ein Problem ist, dass „die Medien“ IT als das Feld einsamer, unsozialer Männer darstellen, die Systeme entwerfen, die anderen schaden. Journalisten und Eltern geben jungen Frauen oft den Rat, „männliche Berufe“ zu meiden. Der überwiegend positive Beitrag von computergestütztem Arbeiten und C++ für die moderne Gesellschaft wird oft vergessen, während Fehlschläge und schädliche Praktiken wie das Hacken im Flutlicht stehen. Die Tatsache, dass computergestütztes Arbeiten oft eine soziale Aktivität ist, die Gruppen von Menschen involviert, die zusammenarbeiten, um in Bereichen wie Bildung, Medizin, Wissenschaft und dem Alltag wichtige Probleme zu lösen, gerät leicht in Vergessenheit. Mich ärgert das Bild vom Softwareentwickler als einem fetten, schlecht gekleideten Nerd, der alleine im Dunklen sitzt, umgeben von leeren Pizzakartons, der massenhaft unverständlichen Text fabriziert. Diese Horrorvision spiegelt nicht die Realität.

Beachtet, dass ich wiederholt „oft“ sage. Natürlich setzt nicht jede Zuschreibung die negativen Bilder fest, aber oft genug richten sie Schaden an. In unserer Wahrnehmung retten heldenhafte Ärzte Menschenleben, nicht das Team der Softwareentwickler und Ingenieure, die erst die grundlegenden Werkzeuge dieser Ärzte und der medizinischen Forschung erstellen, implementieren und am Laufen halten.

Quelle: Bjarne Stroustrup

Mit Studierenden der Shanghai Jiao Tong University (2014)

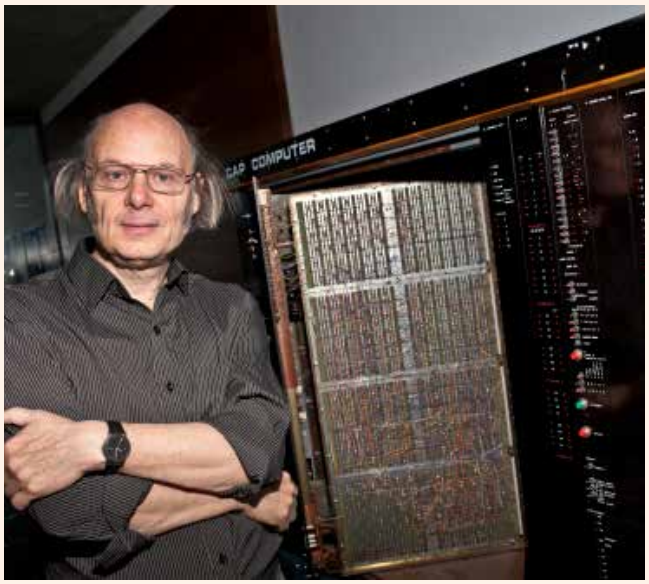
Auf Augenhöhe mit Mathe, Physik und Biologie

Mir scheint, dass der Schulunterricht dem Wert computergestützter Leistungen für die Gesellschaft und der wichtigen Rolle von Entwicklern zu wenig Bedeutung beimisst. Die Betonung liegt zu oft darauf, alles einfach und spaßig zu machen, auf schnellen Erfolgserlebnissen der Studenten. Vielerorts macht sich eine Tendenz bemerkbar, schwierige Themen zu vermeiden, um die Einschreibquote hoch und die Studierenden bei Laune zu halten. An manchen Orten werden Gegenstände unterrichtet, die die Studierenden von einem frühen Zeitpunkt an Grafiken, Bildbearbeitung und Animationen erstellen lassen, ohne ihnen zuerst (oder überhaupt je) das notwendige Wissen zu vermitteln, um zu verstehen, worum es eigentlich geht. Ich habe das bei Highschool-Schülern und auch bei Studierenden der Ingenieurwissenschaften erlebt. Eine Analogie dazu: Wenn man ein Auto bloß fahren will, ist es in Ordnung, den Verbrennungsmotor nicht zu verstehen, aber Ingenieure müssen die Grundzüge davon begreifen, wie eine Maschine funktioniert.

Das rückt C++ ins Hintertreffen im Vergleich zu Programmiersprachen, die – anders als beim Konzipieren ganzer Systeme – für einfachen Anwenderkomfort gemacht sind. Ich wünschte mir, die Schulen würden rechnergestütztes Arbeiten (Computing) als ernsthaftes Thema behandeln, auf Augenhöhe mit Mathe, Physik und Biologie.

Warum schrecken die negativen Klischees weniger Männer ab als Frauen? Ich weiß es nicht. Vermutlich ist ein Teil der Antwort, dass Raketen und Spiele traditionellerweise Männer mehr ansprechen und dass Teile der Gesellschaft Frauen entmutigen und davon abhalten, traditionell männliche Bereiche zu betreten. Ein Teilgrund dafür ist einfach, dass es zu wenig Frauen in IT und Informatik gibt; das ist ein Teufelskreis. In vielen Ländern haben wir es nicht geschafft, eine kritische Menge an Frauen in den IT-Bereich zu bekommen, als Computing noch in den Kinderschuhen steckte.

Quelle: Bjarne Stroustrup



Bjarne Stroustrup in Cambridge vor dem CAP-Computer, mit dem er seine Dissertation erstellt hat (2012)

In der Gruppe kommen Menschen mit Herausforderungen besser klar

Menschen kommen mit den Herausforderungen von Bildung und Ausbildung weit besser klar, wenn sie Teil einer Gruppe von Freunden sind. Und wenn ich meinen Blick über einen Hörsaal mit 250 Studierenden schweifen lasse und darunter nur 20 Frauen erblicke, dann sehe ich ein Riesenproblem. Ich verstehe, warum proportional mehr Frauen als Männer das Studium abbrechen, obwohl sie sich objektiv leichter tun mit dem unterrichteten Material. Meine Absolventenklassen an der Columbia University haben übrigens ein Drittel Frauenanteil, also vielleicht sehen wir gerade Verbesserungen. Ich bin noch optimistisch.

C++ ist ein kleiner Ausschnitt dieses Bildes, aber ich befürchte, dass das C++-Image von „maschinennahem" Code, eingebetteten Systemen, Ingenieursarbeit, Spielen, Finanzen und großen Systemen die Sache schlimmer macht. Wenn wir der Gesellschaft insgesamt und insbesondere Frauen begreiflich machen, wie viel Gutes von profunder, professioneller Softwareentwicklung geschaffen wird, dann wird das das Problem eventuell lösen. Damit ein junger Mensch viele Jahre aufbringt, um ein Feld zu beherrschen, muss dieses Feld in erster Linie als lohnenswert, möglicherweise sogar nobel wahrgenommen werden.

Dranbleiben

Der Schwerpunkt muss darauf liegen, es für Frauen attraktiv zu machen, in diesem Feld zu bleiben. Probleme der Life-Work-Balance können Frauen stärker belasten. Es muss Möglichkeiten sozialen Austauschs geben, der nicht in traditionelle männliche Aktivitäten abdriftet. Es muss breite Unterstützung geben für Schwangerschaften, Mutterschaftsurlaub und Kinderbetreuung. Manche Gesellschaften sind darin weit besser als andere.

Ich habe keine einfachen Erklärungen und Antworten. Es ist mir hier zweifellos nicht gelungen, etwas Grundlegendes zu begreifen oder zu erwähnen. Wie die meisten Menschen finde ich es leichter, Probleme aufzuzeigen, als Lösungen zu empfehlen, aber wir dürfen nicht lockerlassen. Ich bin zuversichtlich, dass wir früher oder später viel mehr Frauen in der Computerwelt und in C++ sehen und dass sie dort glücklich sein werden. Das Feld dafür wird sich verbessern.

Tief verwurzelt in C++: Module, Concepts, Coroutinen

***iX Developer**: Welche neuen Features aus C++20 werden langfristig am meisten Bedeutung für C++ haben und wieso?*

Stroustrup: Module, Concepts und Coroutinen. Module werden uns endlich bessere Codehygiene bringen (weniger Makroverschmutzung), die einfachere Zusammenstellung von Code aus Libraries und viel schnellere Kompilierung. Ich freue mich auf reife Modulimplementierungen, die sich gut in die Umgebung der Programmentwicklung allgemein integrieren lassen. Das wird Zeit brauchen. Bitte seid euch darüber bewusst, dass C++20 jetzt noch einige Monate lang nicht offiziell sein wird, bis das ISO-Sekretariat in Genf seine nicht technische Review abgeschlossen hat. Deshalb sollten wir uns zufriedengeben mit frühen Implementierungen und nicht zu ungeduldig sein bei Unvollkommenheiten. Wir müssen auch erst lernen, Module gut zu nutzen. Vieles von dem, was wir tun, ist durchsetzt von schlechten Angewohnheiten aus fast 50 Jahren von `#include` und `#define`. Nur weil der Präprozessor uns geläufig ist, muss das nicht heißen, dass er auch einfach zu handhaben ist.

Concepts werden wohl generisches Programmieren stark dem annähern, was heute gewöhnliches Programmieren ist. Sie werden unseren Code deutlich flexibler machen, wiederverwendbar und effizient. Als ich die Templates entworfen habe, wollte ich drei Dinge:

Expressiveness Generality, wobei „Expressiveness" für die Fähigkeit steht, eine Idee konzise und direkt auszudrücken. „Generality" ist das Vermögen, das für eine breite Spanne an Themen zu tun, statt nur einen vom Designer so vorgesehenen begrenzten Satz an Auswahlmöglichkeiten zu haben. Unübertroffene Effizienz. Type Checking als „präzise Spezifikation".

Die ersten beiden habe ich hinbekommen, nicht aber den dritten Punkt. Das genügte, um generisches Programmieren und Metaprogrammieren mit Templates zu einem durchschlagenden Erfolg zu machen, aber es strapazierte die Möglichkeiten der Programmiersprache und ihre Implementierung über ein vertretbares Maß hinaus. Der Umstand, dass vernünftige Leute bereit waren, diese Unvollkommenheiten zu ertragen, um die Vorteile zu erlangen, ist ein starkes Argument dafür, dass die grundlegenden Ideen von C++ gesund sind. Also steckten ich und ein paar Freunde von mir es uns als Ziel, auch den dritten Wunsch zu erfüllen, die Spezifikation der Anforderungen eines Templates für seine Argumente. Alexander Stepanov, der Vater des generischen Programmierens im C++-Stil, taufte solche Spezifikationen „Concepts". Sie sind im Grunde nur Prädikatenlogik erster Ordnung, was es einfach macht, sie zu verwenden und über sie nachzudenken, während sie zugleich extrem flexibel sind. Sie gehen nicht zulasten der Runtime. Im Laufe der nächsten paar Jahre werden wir erleben, dass Concepts die Typen und Klassenhierarchien, die bisher hauptsächlich zum Spezifizieren von Funktionsargumenten dienten, langsam ersetzen. Eines Tages werden wir uns wundern, wie wir es jemals ohne sie geschafft haben. Das wird dann so sein, wie wenn wir heute auf sehr frühen C-Code zurückschauen.

Coroutinen waren Teil des frühesten C++

Coroutinen werden uns davon befreien, dort Zustandsfunktionen ausdrücklich zu speichern, wo der nächste Wert von vorherigen Aufrufen abhängt. Solche Berechnungen sind sehr verbreitet im nebenläufigen Programmieren, und ich erwarte, dass Coroutinen den Weg, wie wir viele nebenläufige Systeme organisieren, drastisch vereinfachen. Mit der Einfachheit geht Effizienz in Zeit und Raum einher. Das ist wichtig, weil einige nebenläufige Systeme Hunderttausenden von Aufgaben dienen.

Quelle: Bjarne Stroustrup

Bei einem Lauf in Shanghai (2019)

Coroutinen waren Teil des frühesten C++. In der Tat war die erste C++-Library eine Coroutinen-Bibliothek mit einer gewissen Unterstützung für Simulation im Simula-Stil. Das war grundlegend für die frühe Aufnahme und Verbreitung von C++. Ohne Coroutinen würden wir nicht von modernem C++ sprechen. Leider haben wir sie in den frühen 1990er-Jahren verloren, als komplexere Maschinenarchitekturen es schwieriger machten, Coroutinen gut zu implementieren.

Diese drei Features werden in meinem Buch von 1994, „The Design and Evolution of C++", erwähnt und waren von Anfang an Teil meines Traums von C++ [3]. Ich bin sehr glücklich, einer von denen zu sein, die diesen Traum wahr gemacht haben. Diese drei Merkmale sind nicht nur Ergänzungen zu früheren Eigenschaften, sondern tief in dem, was C++ ist, verwurzelt. Sie beziehen sich auf die Organisation des Codes, das Typensystem und die Verwendung von Hardwareressourcen.

Um mehr über die Ziele, das Design und die Evolution von C++ zu erfahren, empfehle ich meine drei Aufsätze für die „History of Programming Languages"-Konferenzen von ACM SIGPLAN [a, b] und ganz besonders meinen jüngsten Beitrag „Thriving in a Crowded and Changing World: C++ 2006–2020" [c]. Ein riesiges Thema, das Paper dazu ist allerdings keine leichte Kost.

iX Developer: *Was könnten deiner Meinung nach mögliche Hürden sein beim Umstieg auf C++20?*

Stroustrup: Es gibt nur ganz wenige Hindernisse beim Wechsel von C++11, C++14 oder C++17 zu C++20. Die Sprache, die Standardbibliotheken und die großen Implementierungen sind ausgesprochen kompatibel. Aber denkt daran, dass C++20 noch nicht offiziell ist, deshalb sind alle, die die C++20-Switches auf die Implementierungen anwenden, bevor diese Implementierungen volle Unterstützung für C++20 versprechen … all jene gehen als Vorreiter ein gewisses Risiko ein. In der Praxis erfordert das Nutzen der neuen Funktionalitäten eine gewisse Lernbereitschaft und eine Neustrukturierung des Codes, besonders für die Module: Zurzeit verwenden wir Headerdateien und `#include`, in der Zukunft werden wir Module schreiben und Statements importieren.

Jeder Standard ist eine Momentaufnahme

iX Developer: *Welche Neuerungen haben es (noch) nicht in den neuen Standard geschafft und wieso?*

Stroustrup: C++ wächst durch das Feedback von Nutzern und durch den Entwicklungskonsens des ISO-Standardkomitees. Niemand erhält exakt das, was er will, und die Bestandteile brauchen Zeit, um zu reifen. Jeder neue Standard ist bloß eine Momentaufnahme in der Evolution von C++ hin zu einem besseren Werkzeug für Softwareentwicklung in der realen Welt.

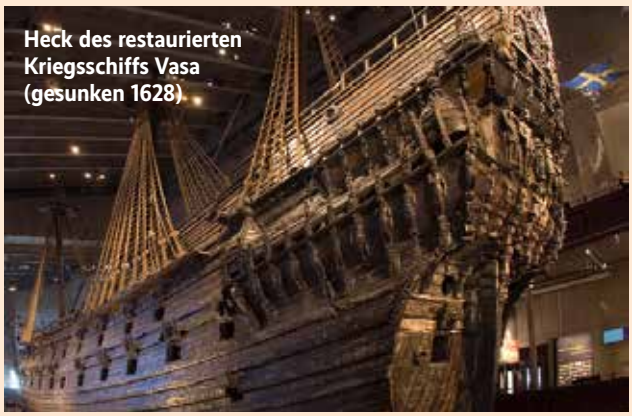
Meine persönlichen Ziele umfassen: einfache Dinge einfach zu machen (ohne komplizierte Dinge unmöglich oder unnötig schwer zu machen), Hardware gut nutzen, Typen- und Ressourcensicherheit, Zero-Overhead Abstraction. Ich bevorzuge Komponenten – ob groß oder klein, Sprache oder Bibliothek – mit Blick darauf, wie gut sie diese Ziele unterstützen und wie sehr sie ihren Nutzern in Relation zum Aufwand dienen, sie einzuführen. Ihr könnt euch über solche Ideen in den regelmäßig erscheinenden Schriften der Lenkungsgruppe des Komitees informieren [d]. Unter vielen anderen Themen findet man darin auch eine Liste mit Verbesserungsvorschlägen.

Ich halte Ausschau nach großen Features, die etwas grundlegend einfacher machen, so wie ein vollständigeres Modell der Concurrency (Executors) [e], und nach Bibliothekskomponenten, die entweder grundlegend sind oder entscheidende Unterstützung liefern für eine bestimmte Art von Anwendung wie etwa Units, Sound und einfache Grafiken. Außerdem halte ich Ausschau nach kleinen Features, die die Notation vereinfachen oder es Programmierern ersparen, einfache, allgemein nützliche Funktionen und Klassen per Hand zu schreiben. Ich bin nicht für eine Vielzahl kleinerer

Remember the Vasa!

We often remind ourselves of the good ship Vasa. It was to be the pride of the Swedish navy and was built to be the biggest and most beautiful battleship ever. Unfortunately, to accommodate enough statues and guns it underwent major redesigns and extension during construction. The result was that it only made it half way across Stockholm harbor before a gust of wind blew it over and it sank killing about 50 people. It has been raised and you can now see it in a museum in Stockholm. It is a beauty to behold – far more beautiful at the time than its unextended first design and far more beautiful today than if it had suffered the usual fate of a 17th century battle ship. But that is no consolation to its designer, builders, and intended users.

Bjarne Stroustrup, How to Write a C++ Language Extension Proposal for ANSI-X3J16/ISO-WG21 (1992), S. 8 [g]



Heck des restaurierten Kriegsschiffs Vasa (gesunken 1628)

Quelle: Karolina Kristensson, the Vasa Museum/SMTM, vasamuseet.se

Komfortmerkmale mit begrenzter Anwendbarkeit und sehe einen stetigen Strom von Vorschlägen für kleine Funktionalitäten, ohne die wir seit 40 Jahren glücklich leben. Jede einzelne würde C++ nur ein wenig mehr Komplexität verleihen, aber insgesamt würden sie ein schreckliches Durcheinander verursachen. Darüber habe ich eine Arbeit geschrieben: „Erinnert euch an die Vasa!“ [f] (s. Kasten „Remember the Vasa!“).

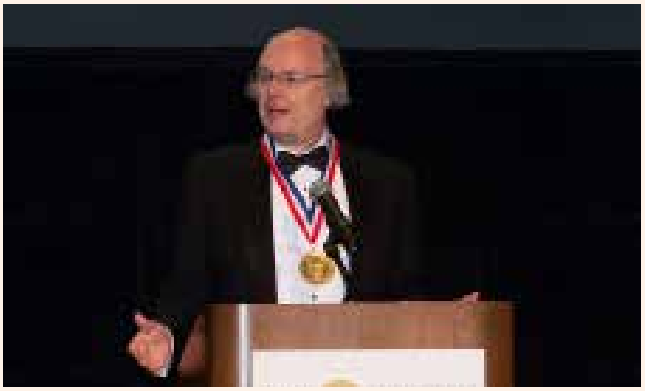
Wir müssen geduldiger sein

iX Developer: *Worauf können Entwickler sich in C++23 freuen?*

Stroustrup: Wir müssen geduldiger sein. COVID-19 verlangsamt die Arbeit des Komitees, so wie es viele andere Arbeiten drosselt. Ich hoffe auf Problembehebungen, ein paar Minor Features, „um C++20 abzurunden“, und vielleicht auf einen oder mehrere der folgenden Punkte aus dem Prä-Pandemie-Plan des Komitees für C++23: Standard-Library-Module, Bibliotheksunterstützung für Coroutinen, Unterstützung der Networking-Bibliothek, Executors, funktionales Programmieren im Pattern-Matching-Stil, Static Reflection [h]. [Anm. Red.: s. hier im Heft „Zukunftsmusik“, Seite 92.]

iX Developer: *Was sind deine persönlichen Wünsche für die Zukunft – nicht nur, aber auch für die Entwicklung von C++?*

Stroustrup: Wie die meisten Leute habe ich eine lange Liste davon, was ich gern verbessert sähe in der Welt, aber ich möchte mich hier lieber nicht tiefer über Politik auslassen, abgesehen von der Hoffnung auf einen sicheren, wirksamen und günstigen Impfstoff gegen COVID-19 und dass die Menschheit beim Klimawandel die Kurve kriegt.



Quelle: Bjarne Stroustrup

Danksagung für den Draper-Preis der National Academy of Engineering, Washington, D.C. (2018)

Nun zurück zu den Themen, bei denen ich tatsächlich einige Einblicke und möglicherweise sogar einen gewissen Einfluss habe. Für C++ möchte ich garantiert typen- und ressourcensicheren Code. Ich bin Teil eines Projekts, um das zu bewerkstelligen. Wenn man in der Lage ist, willkürlich komplexen C++-Code zu schreiben, hat man noch lange keine Garantien. Es ist möglich, in C und C++ Code zu schreiben, dessen Richtigkeit sich nicht beweisen lässt. Wir haben es da mit Gödels Unvollständigkeitssatz zu tun [i]: Um Garantien zu erhalten, müssen wir die Leute daran hindern, zu komplexen Code zu schreiben.

Guidelines für modernes C++

Jedenfalls können wir Regeln für den Gebrauch von C++ schreiben, die sich von einem Static Analyzer überprüfen lassen. Das Projekt heißt „The C++ Core Guidelines“ und ermutigt zu einem modernen Stil von C++, den wir überprüfen können, um Speicherbeschädigungen, Speicherlecks und Typenverletzung zu eliminieren, ohne einen Garbage Collector einzuführen oder uns kostspielige Laufzeitüberprüfungen aufzuerlegen [j]. Das beruht auf dem Prinzip der Untermenge (Subset) einer Obermenge (Superset): Zuerst erweitert man die Sprache um grundlegende Bibliotheken, und dann kann man Bestandteile verbannen, die Probleme verursachen. Dieser Ansatz ist notwendig, weil die gefährlichen Features oft genau diejenigen sind, die du zum Aufbau der Basisbibliotheken benötigst. Eine statische Überprüfung ist erforderlich, um Regelverstöße aufzudecken, die zu Fehlern führen könnten. Derzeit wird Visual Studio C++ mit einem Checker ausgeliefert, und Clang tidy führt einige Prüfungen durch. Meine Hoffnung ist, dass sich so etwas wie diese Regeln sehr weit verbreiten wird und dass Prüftools standardmäßig in allen wichtigen C++-Implementierungen verfügbar sein werden. Ich hoffe, dass dies bald geschieht, denn es würde so viele weitverbreitete und unnötige Probleme beseitigen.

Das Gespräch führten Silke Hahn (sih@ix.de) und Madeleine Domogalla (mdo@ix.de), Redakteurinnen von heise Developer.

Literatur

[1] Bjarne Stroustrup; Programming; Principles and Practice Using C++; Addison-Wesley 2008 (2. Aufl. 2014) (dt.: Einführung in die Programmierung mit C++)

[2] Bjarne Stroustrup; A Tour of C++; Addison-Wesley 2013 (2. Auflage 2018) (dt.: Eine Tour durch C++)

[3] Bjarne Stroustrup; The Design and Evolution of C++; Addison-Wesley 1994

Onlinequellen

[a] Bjarne Stroustrup; A History of C++: 1979–1991 (1993); in: Proceedings of the Second ACM SIGPLAN Conference on History of Programming Languages; Association for Computing Machinery, S. 271–297: doi.org/10.1145/154766.155375

[b] Bjarne Stroustrup; Evolving a Language in and for the Real World: C++ 1991–2006 (2007); in: Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages; Association for Computing Machinery, S. 1–59: doi.org/10.1145/1238844.1238848

[c] Bjarne Stroustrup; Thriving in a Crowded and Changing World: C++ 2006–2020 (2020); in: Proceedings of the Fourth ACM SIGPLAN Conference on History of Programming Languages; Association for Computing Machinery: doi.org/10.1145/3386320

[d] Howard Hinnant, Roger Orr, Bjarne Stroustrup, David Vandevoorde, Michael Wong; Direction for ISO C++ (2020): open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2000r2.pdf

[e] Sergei Murzin, Michael Park, David Sankel, Dan Sarginson; Programming Language C++ Evolution – Pattern Matching (2020): open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1371r2.pdf

[f] Bjarne Stroustrup; Remember the Vasa! (2018): stroustrup.com/P0977-remember-the-vasa.pdf

[g] Bjarne Stroustrup, How to Write a C++ Language Extension Proposal für ANSI-X16/ISO-WG21 (1992) S. 8: stroustrup.com/how-to-write-a-proposal.pdf

[h] Ville Voutilainen; To Boldly Suggest an Overall Plan for C++23 (2019): open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0592r1.html

[i] Gödelscher Unvollständigkeitssatz: en.wikipedia.org/wiki/Gödel's_incompleteness_theorems

[j] C++ Core Guidelines: github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md

[k] Danksagung für den Draper-Preis der National Academy of Engineering (NAE), Washington, D.C. (2018) bit.ly/352ZDy3