The problem with programming

Bjarne Stroustrup is talking about his views on programming. The questions of the interview are in mixed order. Match questions and answers.

1. Why is most software so bad?

2. How can we fix the mess we are in?

3. The idea behind C++ was that programmers would work harder in return for more efficient code. Today there are a lot of software developers and computers are very fast. Does this vitiate the point of C++?

4. In retrospect, in designing C++, wasn't your decision to trade off programmer efficiency, security, and software reliability for run time performance a fundamental mistake?

5. What do you regret the most?

6. How do you account for the fact that C++ is both widely criticized and resented by many programmers but at the same time very broadly used? Why is it so successful?

a) No regrets! Well, of course I dream of what I might have done differently and better, but seriously, who am I to second-guess, say 1984 vintage Bjarne? He may have been less experienced than I, but he was no less smart, probably smarter, and he had a better understanding of the world of 1984 than I have. C++ has been used to build many systems that enhance our lives, and it has been a significant positive influence on later languages and systems. That's something to be proud of.

b) C++ has indeed become too "expert friendly" at a time where the degree of effective formal education of the average software developer has declined. However, the solution is not to dumb down the programming languages but to use a variety of programming languages and to educate more experts. There have to be languages for those experts to use - C++ is one of those languages.

c) Some software is actually pretty good by any standards. Think of the Mars Rovers, Google, and the Human Genome Project. That's quality software! On the other hand, looking at "average" pieces of code can make me cry. The structure is appalling, and the programmers clearly didn't think deeply about correctness, algorithms, data structures, or maintainability. I think the real problem is that we" (that is, we software developers) are in a permanent state of emergency, grasping at straws to get our work done. We perform many minor miracles through trial and error, excessive use of brute force, and lots and lots of testing, but - so often - it's not enough.

d) In theory, the answer is simple: educate our software developers better, use more-appropriate design methods, and design for flexibility and for the long haul. Reward correct, solid, and safe systems. Punish sloppiness. In reality, that's impossible. People reward developers who deliver software that is cheap, buggy, and first. That's because people want fancy new gadgets now. They don't want inconvenience, don't want to learn new ways of interacting with their computers, don't want delays in delivery, and don't want to pay extra for quality.

e) The glib answer is, there are just two kinds of languages: the ones everybody complains about and the ones nobody uses. The purpose of a programming language is to help build good systems, where "good" can be defined in many ways. My brief definition is, correct, maintainable, and adequately fast. Aesthetics matter, but first and foremost a language must be useful; it must allow real-world programmers to express real-world ideas succinctly and affordably. The main reason for C++'s success is simply that it meets its limited design aims: it can express a huge range of ideas directly and efficiently. C++ was not designed to do just one thing really well or to prevent people doing things considered "bad." Instead, I concentrated on generality and performance.

f) Well, I don't think I made such a trade-off. I want elegant and efficient code. Sometimes I get it. These dichotomies (between efficiency versus correctness, efficiency versus programmer time, efficiency versus high-level, et cetera.) are bogus. What I did do was to design C++ as first of all a systems programming language: I wanted to be able to write device drivers, embedded systems, and other code that needed to use hardware directly. Next, I wanted C++ to be a good language for designing tools. That required flexibility and performance, but also the ability to express elegant interfaces. My view was that to do higher-level stuff, to build complete applications, you first needed to buy, build, or borrow libraries providing appropriate abstractions. Often, when people have trouble with C++, the real problem is that they don't have appropriate libraries -or that they can't find the libraries that are available.

Study the interview with Bjarne Stroustrup and answer the following questions:

1. What does Bjarne Stroustrup think of modern software?
2. What does he criticize about some programmers? Why does he think this is the case?
3. What does he think should be done to improve the quality of programming work? Why, in his opinion, isn't it done?
4. Does he think C++ is still a relevant programming language? For whom?
5. Does he feel he compromised on the design of C++? Why or why not?
6. What aims did he have in mind when designing C++?
7. What does he think is the underlying problem many people have with C++?
8. What comprises a "good" programming language, in his opinion?
9. Why does he think C++ is so successful?
10. Is Bjarne Stroustrup 100% satisfied with his achievements?

In the Interview Bjarne Stroustrup uses a number of colloquial expressions. How would you translate or explain the following terms in German?

1. to fix the mess
2. to trade off …for
3. How do you account for
4. is this a joke
5. glib answer
6. first and foremost
7. “the crowd”
8. grasping at straws
9. brute force
10. sloppiness
11. buggy
12. 1984 vintage
13. to dumb down
14. for the long haul

Bjarne Stroustrup makes some pretty strong statements. Discuss with a partner whether you agree wholeheartedly, in part or not at all with the following. Report back to the group.

- Software developers are not as well-trained as they used to be.
- There should be special programming languages just for experts.
- The average software is poorly programmed.
- Programmers are put under pressure to deliver the goods as quickly as possible.
- People just want quick, easy solutions to their computer needs.
- Programmers often don't make efficient use of libraries.
- There is no basic contradiction between efficiency versus correctness or efficiency versus time invested regarding programming work.

Complete the following sentences with the correct expressions from the box.

edit • compilation • construct • devices • designed • create
•embedded • environment • execution • modelling • oriented • practices
• predefined • purpose • represents • running • systems • toolbox

The programming language C is an "object 1 _________ programming language". Turbo Pascal allows you to 2 ___________the code while 3 _____________the compiler. Java is 4 _____________for programming small electronic 5 _________ and to 6 ______________ web pages. Simpler in design and use is Javascript which is 7 _____________ in HTML documents. Visual Basic is a programming 8________________ which uses the language BASIC and 9 __________ objects chosen from a 10 _____________to write general 11 _____________ programs for Window applications. Delphi is a powerful, object oriented Windows and Linux development tool with very fast 12 _____________________________ and 13 ____________________________ Unified 14 ____________________ Language (UML) is used to 15 ___________and document the artifacts of software 16________________. UML 17 ______________ the best engineering 18________________for the modelling of large and complex systems.

Acronyms

Often confusing but admittedly convenient at times, acronyms and abbreviations abound in the computer world. Do you know what all the letters in the ones below stand for? With a partner see how many you can identify correctly.

WLAN • ALU • BASIC • BIOS • CAD • CAM • CAE • CD • CD-ROM • CPU • bps • DTP • DVD • FAQ • CGI • FTP • GB • GUI • HTML • Hz • HTTP • I/F • I /O • IP • IRC • DSL • ISP • IT • LAN • MB • MP • OCR • OS- PL/1 • PPP • RAM • TCP/ IP • URL • WAN • WWW • PIN • VGA • VR • ppi

Read the text and find the English equivalents for the given German terms.

Programming languages

A programming language consists of a set of vocabulary and grammar rules for instructing a computer on how to perform specific tasks. The term usually refers to high-level languages, such as BASIC, 5 C, C++, COBOL, FORTRAN, ADA, and Pascal.

Each language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions. High-level programming languages are more complex than the languages the computer actually understands, called machine languages. Each different type of CPU has its own unique machine language. Lying between machine languages and high-level languages are languages called assembly languages.

Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers. Machine languages consist of numbers only. Lying above high-level languages are languages called fourth-generation languages (usually abbreviated 4GL). 4GLs are far removed from machine languages and represent the class of computer languages closest to human languages.

German	English
einen Computer anweisen	
Aufgaben ausführen	
bezieht sich auf	
einzigartig	
tatsächlich verstehen	
leichter zu programmieren	
Namen durch Zahlen ersetzen	
nur aus Zahlen bestehen	
abgekürtz 4GL	
weit entfernt von	