

When you talk about productivity there is always the ghost of Frederick Taylor hanging around. One of the goals of Taylorism was to replace expensive (and slow) skilled craftsmen by unskilled workers and yet produce quality goods. It's rarely said in the software area, but the same concern is still there.


Productivity

Can it be done by cheap beginners rather than expensive craftsmen?



A time-line of Software Engineering

I have tried to set up a time-line of IT with (in blue) the progresses in the field of software engineering. Needs changed as software slowly became an industry.



Initials needs were modest.

1950

19

assembly language

19

Computers were big expensive machines that only very big companies could afford.

compilers/interpreters

libraries

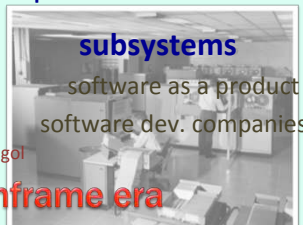
subsystems

software as a product

software dev. companies

Cobol + Fortran + Algol

Mainframe era



The idea of software as a product distinct from hardware only appeared in the 1960s

1955

1960

19

Project Management

Portability

Beginning of OOP

Beginning of AI

Cobol + Fortran

Lisp

Databases

PL/1

Transactional Monitors

C

Unix

First virtualized system

Minicomputer era

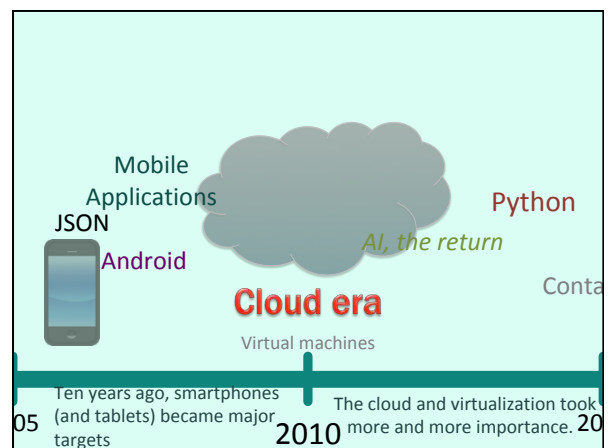
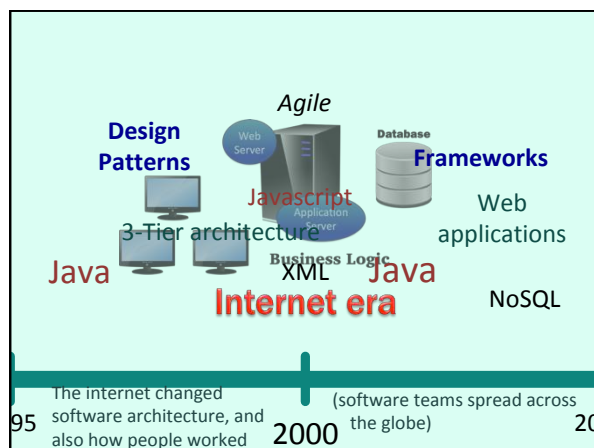
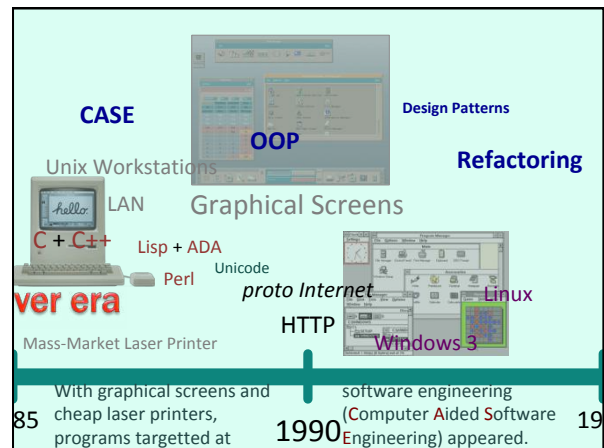
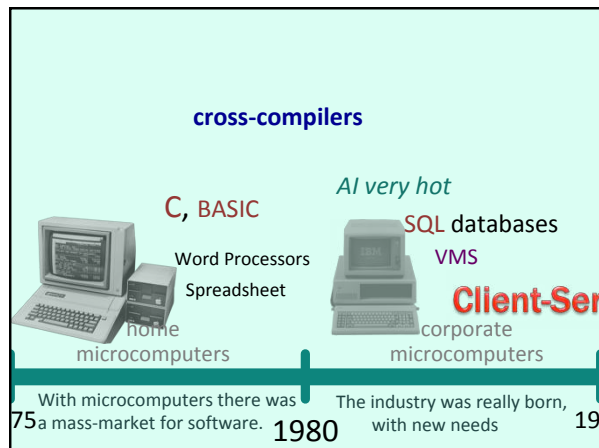
early microcomputers

It all changed in the 1970s when computers became more affordable

1965

1970

19



It's very hard to predict where we are going. What is certain is that in the same way that software and hardware used to be closely linked and people were not considering software independently, first there has been a separation between upper and lower software layers (every piece of hardware actually contains a lot of software), and then hardware is becoming more and more abstract. It's likely that in the near future a lot of people currently working for companies will move to "infrastructure companies" providing raw CPU and storage. That will probably create new needs in application software, because you cannot (if you want decent performance) code exactly in the same way when your application runs locally or on a distant server. Software always needs more quality (failures have serious impact) – and is written by not always highly skilled people.

15 2020 20

What happened historically?

010010011111000100010000111000

93C4438

1 0 0 1 0 0 1 1

Since the beginnings, computer memories have held nothing but 0s and 1s (it might change one day with quantum computers). The first step in "software engineering" was to use hexadecimal (base 16) encoding to replace sequences of four bits by one digit between 0 and 9 or a letter between A and F. Fewer errors.

What happened historically?

Macro Language

| | | | |
|---------|----------|-----------|---|
| CLI | SEX,C'M' | Male? | |
| | BNE | IS_FEM | If not female, go on |
| | L | 7,MALES | Load current value of MALES into register 7 |
| | AL | 7,=F'1' | add 1 |
| | ST | 7,MALES | store back the result |
| | B | GO_ON | Finished with this portion |
| IS_FEM | EQU | * | A label |
| | L | 7,FEMALES | Load current value in FEMALES into register 7 |
| | AL | 7,=F'1' | add 1 |
| | ST | 7,FEMALES | store back the result |
| GO_ON | EQU | * | - rest of program - |
| * | | | |
| MALES | DC | F'0' | Counter for MALES (initially=0) |
| FEMALES | DC | F'0' | Counter for FEMALES (initially=0) |

Very quickly appeared programs translating simple keywords into the correct sequence of bits.

processor specific

But you had to completely rewrite everything to run a similar program on another computer.

What happened historically?

The next step was with high-level languages and compilers. The compiler was computer-specific, but not the program. Around the same time appeared Grace Hopper's COBOL (Common Business Oriented Language) ...



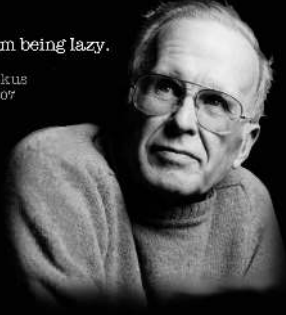
Grace Hopper (1906-1992)

What happened historically?

Much of my work has come from being lazy.

John Backus
1924-2007

.. and Backus' FORTRAN (FORmula TRANslator) for scientific applications. With FORTRAN in particular a lot of scientific libraries were written, implementing frequent computations (for instance on matrixes, or equation-solving). Reusable software.



Functions/Libraries

How can we design the code for reuse?

Designing reusable software (functions that you can use in very different contexts) is like designing engines that can be used in many cars, and it's far from trivial. I'm going to illustrate it with a C example.

Example: Sort routine

```
void my_sort(int values[])
```

Because most applications are basically applying processes to data held in memory, data storage has always been important, and fast data retrieval particularly important. To retrieve data, it's better to keep it sorted, and sorting efficiently has always been an important concern. You can imagine writing a function to sort an array of integers. In C, you have no way to know how many values you have in an array. You could imagine having a "flag" values (say -1) that indicates the end of data, but then it means that -1 is a forbidden data value. Not very generic.

Example: Sort routine

```
void my_sort(int values[], int count)
```

A better solution is to pass the number of items alongside the array (this is exactly how the `main(int argc, char *argv[])` of a C program receives command-line arguments from the operating system). This routine could sort basically any array on integers – but it can only sort integers. That may not be generic enough to justify inclusion into a library.

Example: Sort routine

By contrast, here is the `qsort()` function available in C. You give it "a pointer", how many values have to be sorted, the size (width) of each element in the array, and even the function that decides which of two elements is greater than the other.

```
void qsort(void *base, size_t nel,
          size_t width,
          int (*compar)(const void *,
                       const void *));
```

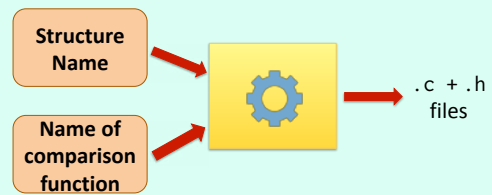
Casts everywhere!

Lots of responsibility with the developer

A good developer can use this. A beginner will have a lot of trouble, and probably crash the program even if it compiles.

One possible approach:

Code generator



One possible approach would be to say "let's write a program that takes as input basic information and generates correct code".

Better approach:

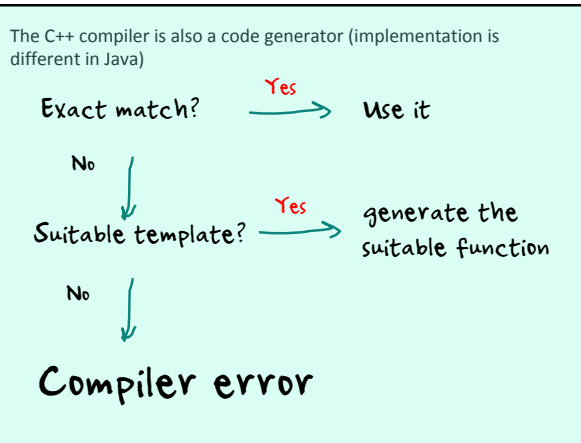
Built in compiler <GENERIC TYPE>

C++

When Stoutstrup designed C++, he actually took that same idea and built it into the compiler, with the introduction of generic types. Generic types work because of another feature introduced in C++ (and also available in Java) known as overloading. In C functions are identified by their name only. In C++ or Java, several methods can have the same name if the compiler can distinguish them by the type and/or number of arguments.

Instead of writing the actual function, you actually write a template with a generic type. This template is used by the compiler to generate actual functions.

```
template<class T> float average(T *arr, int n) {
    float total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total / n;
}
```



Like C++, Java wants strong typing – you must declare exactly what is the type of every variable you use, and the compiler checks that everything is consistent to avoid errors.

Java = strong typing

SAFE

but ...

```
String[] months = {"Jan", "Feb", "Mar", "Apr",
                  "May", "Jun", "Jul", "Aug",
                  "Sep", "Oct", "Nov", "Dec"};
int[] sales = {120, 98, 75, 110, 150,
               180, 170, 174};
```

```
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
120 98 75 110 150 180 170 174
```

The requirement for strong typing may result in too much effort. Suppose you want to display the contents of an array of strings and of an array of integer values.

```
public static void displayArr(String[] arr) {
    int n = arr.length;

    for (int i = 0; i < n; i++) {
        if (i > 0) {
            System.out.print("\t");
        }
        System.out.print(arr[i]);
    }
    System.out.println("");
}
```

The code will be exactly the same for both arrays. The only thing that will change will be the parameter type.


```
public static <T> void displayArray(T[] arr) {
    int n = arr.length;

    for (int i = 0; i < n; i++) {
        if (i > 0) {
            System.out.print("\t");
        }
        System.out.print(arr[i]);
    }
    System.out.println("");
}
```

With objects, you can write a generic function that takes an arbitrary type (in other words, class) name, and it will work with any object.

displayArray(months);
displayArray(sales);

```
public static <T> void displayArray(T[] arr) {
```

Before the return type (it could be T)

<T>

<T,U>

The generic type is indicated between angular brackets at the beginning, before the return type. You can use several generic types in a function (that could ultimately be the same, or not).

Java requires objects because, contrary to C++, it doesn't generate code – it just adds casts (which means that it specifies that this reference is a reference to an object of a specific class) and checks that everything is consistent.

Generic methods can be overloaded

```
public static <T> void displayArray(T[] arr)

public static <T> void displayArray(T[] arr,
                                   String sep)

public static void displayArray(Date[] arr,
                                   String format)
```

Contrary to C++, a generic Java function is real code, not a template. It can be overloaded, by another generic function, or by a class-specific function.

Exact Match?

YES

Use it

NO

Generic Match?

YES

Use it

NO

Error

Otherwise, the algorithm that allows the Java compiler to decide what to use is the same, minus code-generation, as in C++.


```

public class Boxing1 {

    public static void displayVal(Integer val) {
        System.out.println(val);
    }

    public static void main(String[] args) {
        int n = 10;

        displayVal(n);
    }
}

```

Works OK

You see here a function that expects an object (a reference), to which a value is passed. The compiler takes the address of the value and passes it to the function.

```

public class Boxing2 {

    public static void displayVal(Integer[] valarr) {
        for (int i = 0; i < valarr.length; i++) {
            System.out.println(valarr[i]);
        }
    }

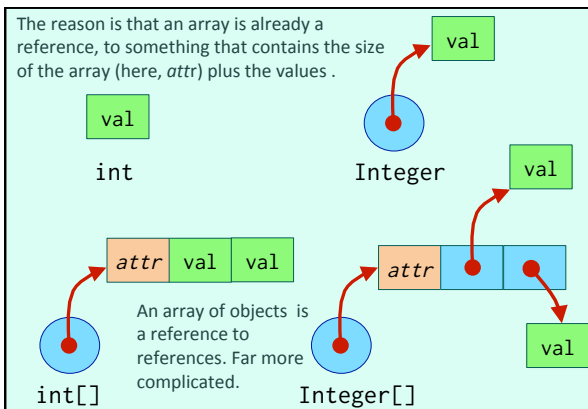
    public static void main(String[] args) {
        int[] n = new int[3];

        for (int i = 0; i < 3; i++) {
            n[i] = i;
        }
        displayVal(n);
    }
}

```

If you try to compile this program (that doesn't look very different) it will fail.

The reason is that an array is already a reference, to something that contains the size of the array (here, *attr*) plus the values .



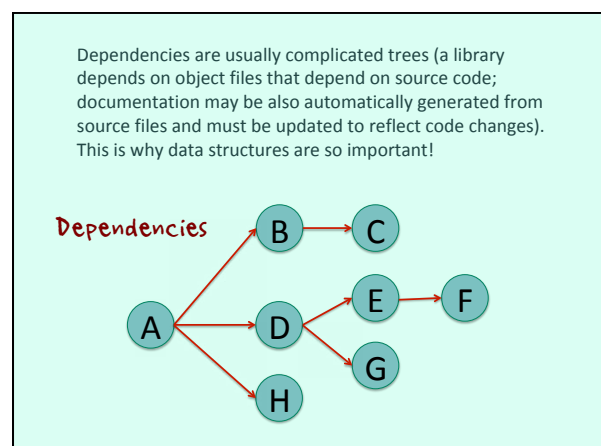
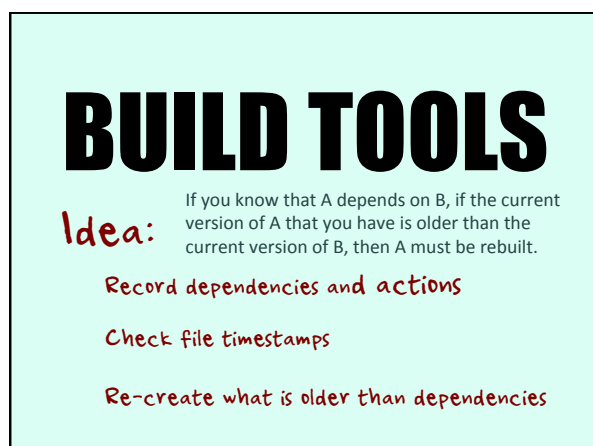
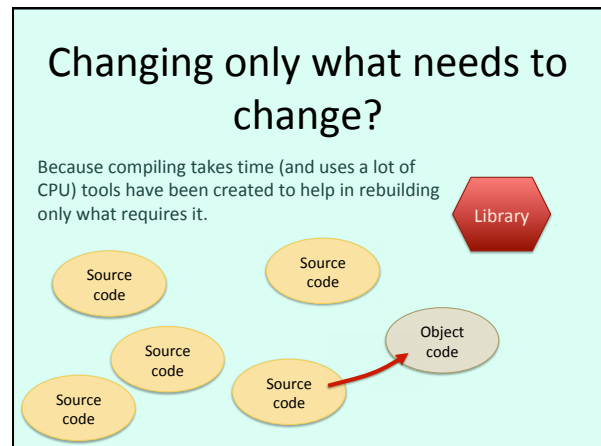
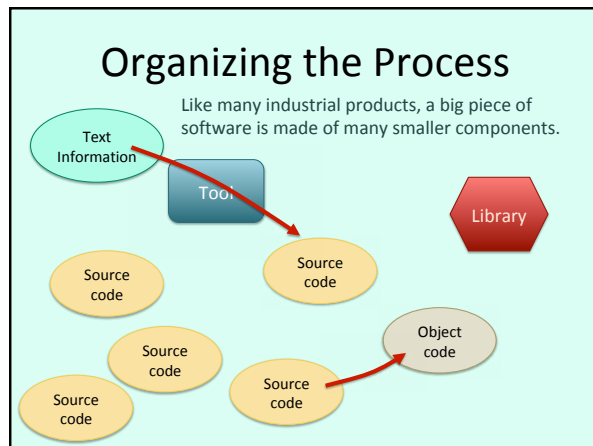
Arrays of Objects

```

Obj[] o = new Obj[size];
int count = 0;
o[count] = new Obj(...);
count++;

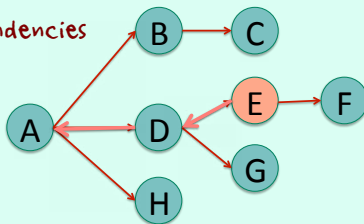
```

Instantiating an array of objects only reserves memory for storing references. Each object in the array must be instantiated in turn.



If E is modified, you see that D depends on it, and A on D. By automatically applying the rules that say how to generate D from E, then A from D, you automatically update A without having to touch what was left unchanged. One of the first build tools was 'make'. In the Java world, you also find 'Ants' or (more often) 'Maven'.

Dependencies



TREES of dependencies

Recursion

Who says "trees" means "recursion", and here is the rub. There is a huge need for software developers, many of them have trouble with recursion and with pointers. One of the goal of the Java creators was to hide the difficult parts to enable more (and cheaper) people to code.

“ If I may be so brash, it has been my humble experience that there are two things traditionally taught in universities as a part of a computer science curriculum which many people just never really fully comprehend: pointers and recursion. ”



Joel Spolsky

Two Problems

- 1 The average programmer doesn't understand pointers and recursion too well

"Genericity" is also very important, because "Generic" code means code that you can reuse (less work) and also code that has already been tested (very important!)

- 2 Genericity