This was the actual talk
on Data Reliability
given at the conference

Slide 1

① Here is an escample of a flowchart. ①
And here I must already apologise for ~~living~~ my life in
an ivory tower, ~~and~~ ~~showing~~ which lead giving you
such a small escample. ~~Please believe that~~
~~my remarks will apply~~ ~~two~~ also to the very much
real life wall-to-wall and f
~~larger examples which are sometimes found to~~
~~occupy the walls of a programming managers office~~ —
I believe that
~~And they will apply with even greater force~~
We all know that ~~a~~ real life flowchart ~~stretches~~
for a real life program stretches from wall to wall, indeed
from floor to ceiling. Allow me to ~~postulate~~ assure you that the
following ~~remarks~~ problems arise apply to larger cases, ~~in~~ in fact with even greater severity

②

② 1. We cannot conveniently input a flowchart to a computer

2. Nor can we conveniently obtain it as output

3. It does not decompose readily into parts

it is intellectually unmanageable

4. A Local fault~~s~~ (in an arrow has) ~~have~~ global consequences

5. Dynamic structure change causes global consequences.

6. Non-local jumps are timeconsuming.

Slide 2. And here is a data diagram, a picture of a ~~7. Too much paper — too many pages.~~ supposed data structure. It suffers from all the same disadvantages. rep.

But an overwhelming

③

③ Too much paper — too many pages

A picture is worth a thousand words,

but not when it stretches over a thousand pages.

and especially when the most interesting part of the picture is the words ~~written above in the boxes.~~ ~~in the boxes~~ that it contains.

An educated man or woman is one who has learnt to appreciate the conciseness and expressiveness of continuous prose, 4

4 It is only In ~~Flowcharts~~ that we ~~are~~ the picture ~~children's~~ comics of our childhood must every word must be ~~surrounded~~ by a balloon. ~~Let~~ us But ~~when we grow up,~~ we learn to appreciate ~~the power and~~ conciseness ~~and expressiveness of the written~~ word. ~~continuous~~ prose.

slide 3.

(5) Now I would like to survey with you the basic structuring methods for data, and point out the close analogy with structuring methods for programs. In each case I will show how the traditional voluminous picture can be replaced by a concise and expressive (and vigorous) notation. slide 4 The simplest and most important method is the direct or cartesian product, which corresponds closely to the concatenation or composition of statements in a programming ~~language~~ On ~~the~~ (message on the) left of the slide in black ~~you~~ ~~see an~~ ~~ex~~ ~~imagine~~ ~~excerpt~~ should remind you of a text from your favourite gospel on structured programming, ~~On the right of the slide in blue shows~~ I always used to draw those boxes one under the other; until I realised that for the purposes of reliable software it was better to draw them as a cascade.
On the right ...

5 Conditional / union
6. Iteration / sequence.
7. Recursion.
7a Recursion.

The next data structuring tool, ~~like its~~ is recursion. Like recursion in program structuring, it is not often required; but when it is required, it is required rather badly. Unfortunately I had a bit of trouble drawing a picture of recursion. slide 7, which must of course, be a picture of a picture of a picture of a picture of a picture... On this slide I have tried my best to ~~draw~~ illustrate my final ~~thousand~~ answer to the old saying of Confucius — ~~here is~~ I offer you the word that worth a thousand pictures.

But I fear that the controversy that has surrounded the avoidance or non-avoidance of jumps has ~~be~~ obscured one vital fact. It is not by avoiding jumps that one obtains well-structured programs. — indeed I fear ~~[illegible]~~ that many jumpfree programs have a lousy structure. The ~~truth~~ fact is that by writing well-structured programs we avoid the use of jumps — indeed I am glad to say that ~~all wish to~~ eventually ~~even~~ all desire to jump will wither away, ~~and we attain a state of [illegible]~~ The absence of jumps is ~~the system~~ symptom, not the cause, of structured programming, ~~not~~ it is the outward and visible sign of an inward and spiritual grace

~~So there remains the central question, what is whether the central issue in the structuring of programs, and how does it contribute to accuracy and reliability? I think it lies in what Dijkstra calls a "separation of concerns," the successful decision to embody one group of related concerns in a compact~~

And the essence of this grace is ~~the same as~~ very similar to my mula for data ~~structuring~~ the conscious attempt to program in one place all the code arising from one programming decision, and to keep that code ~~of the~~ of as separate as possible from that which embodies a different decision

And one of the most successful methods of separating our concerns is the conscious use of abstraction and its separation from the mass of highly relevant detail which is used to implement it. slide 8 shows an early example of ~~one of the few~~ an abstract program for printing all primes up to a million. On a concrete machine ~~which~~ for which the test of primeness and printing are ~~built in~~ available ~~as~~ instructions, this would be a concrete program. Since such a machine does not exist, we must construct it by ~~a~~ writing detailed programs to implement it. But at all costs we must ~~separate~~ keep these detailed concerns separate from the abstract program which they are designed to ~~imp~~ support, and use the abstract program as a guide and framework with which to organise the details. And if anyone is discontented with this small example, he may be assured that the advice I give will be even more useful for large programs, as he may ~~not~~ readily convince himself by replacing the constant million by a trillion or a quintillion before embarking on implementation.

slide 9 shows an example of an abstract data structure.

explain

no details of representation

In this talk, I have had occasion frequently to apologise for the simplicity of my examples.
I~~n conclusion~~, But I will ~~[illegible]~~ offer
no apology for the simplicity of the methods
which I have described.
~~I advocate~~, ~~nor for the simplicity of~~
~~the examples which I have used to illustrate~~ remarks apply and with even greater force to the largest applications
~~them,~~ ~~Because I believe that my~~ ~~As~~ ~~In my belief~~ I believe that a
~~for the solution of toy pro~~ complex battery of
elaborate features and facilities may be
~~acceptable or even~~ very attractive in the solution
of the simple ~~toy~~ and small example problems that appear in the
~~software~~ ~~software~~ manuals and sales literature of our
leading purveyors of ~~programming languages and~~ data base management systems
but for really large and complex problems, we
should ~~need~~ prefer to confine ourselves only the simplest methods. Simplicity is
the unavoidable price which we must pay for
reliability. Let us hope we do not find it
too high a price. ~~to pay~~.

## 11. References and Pointers.

One remarkable feature of the structuring methods introduced here is that they make no mention of the reference or pointer, which are traditionally regarded as the prime means of structuring data. In this respect, references seem similar to go to statements, which have traditionally played a major role in computer programming, and which seem to be going rapidly out of fashion. In fact the analogy goes deeper. In the implementation of data structures use may be made of machine addresses, just as jumps are used in machine code to implement conditionals, and while loops and procedures. The major structuring disadvantage of the jump is that it creates new wide interfaces between distant parts of a program, which look as though they should be separate, and the slightest change to a program can propagate errors rapidly and uncontrollably along these interfaces. I suspect that the same is true of a reference, pointing from one part of a data structure to another distant part, which ought to be disjoint. And I expect that there will be yet another analogy - the recommendation to remove references from data structuring will meet as much controversy as that to remove go to's from programming; and perhaps even more so, because it runs counter to the still prevalent belief in integrated information systems, relational data bases, etc.; and suggests that it may be preferable to go back to earlier, simpler, techniques using separate files without cross references, and holding in one place all data relating to each single item of information.