AALBORG UNIVERSITY

Perspectives on Software Engineering

Peter Dolog
dolog [at] cs [dot] aau [dot] dk
E2-201
Information Systems
February 8, 2007

Concepts (Warm Up)

Software
Software Engineering
Vs. Computer Science
Vs. System Engineering
Software Process
Software Process Model
Software Development Costs
Software Engineering Methods
CASE
Software Attributes

What is SE?

WHAT IS SOFTWARE ENGINEERING?
The IEEE Computer Society defines software engineering as "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)."

SWEBOK - Knowledge Categories

<table>
<tr><td rowspan="2">Specialized
Practices used only for certain types of software</td><td>Generally Accepted
Established traditional practices recommended by many organizations</td></tr>
<tr><td>Advanced and Research
Innovative practices tested and used only by some organizations and concepts still being developed and tested in research organizations</td></tr>
</table>

Figure 1 Categories of knowledge

SOE - Related Disciplines

Zelkowitz (1978):

Mathematics	Engineering	Management Science
Algorithms	Costs and Tradeoffs	Requirements, Risks, Personnel, Monitoring

SWEBOK (2004):

Table 2 Related disciplines

◆ Computer engineering	◆ Project management
◆ Computer science	◆ Quality management
◆ Management	◆ Software ergonomics
◆ Mathematics	◆ Systems engineering

SD Life Cycle (Zelkowitz)

Requirements analysis
Specification
Design
Coding
Testing
Operation and maintenance

Nato-seminar in Garmisch 1968

NATO SOFTWARE ENGINEERING CONFERENCE 1968

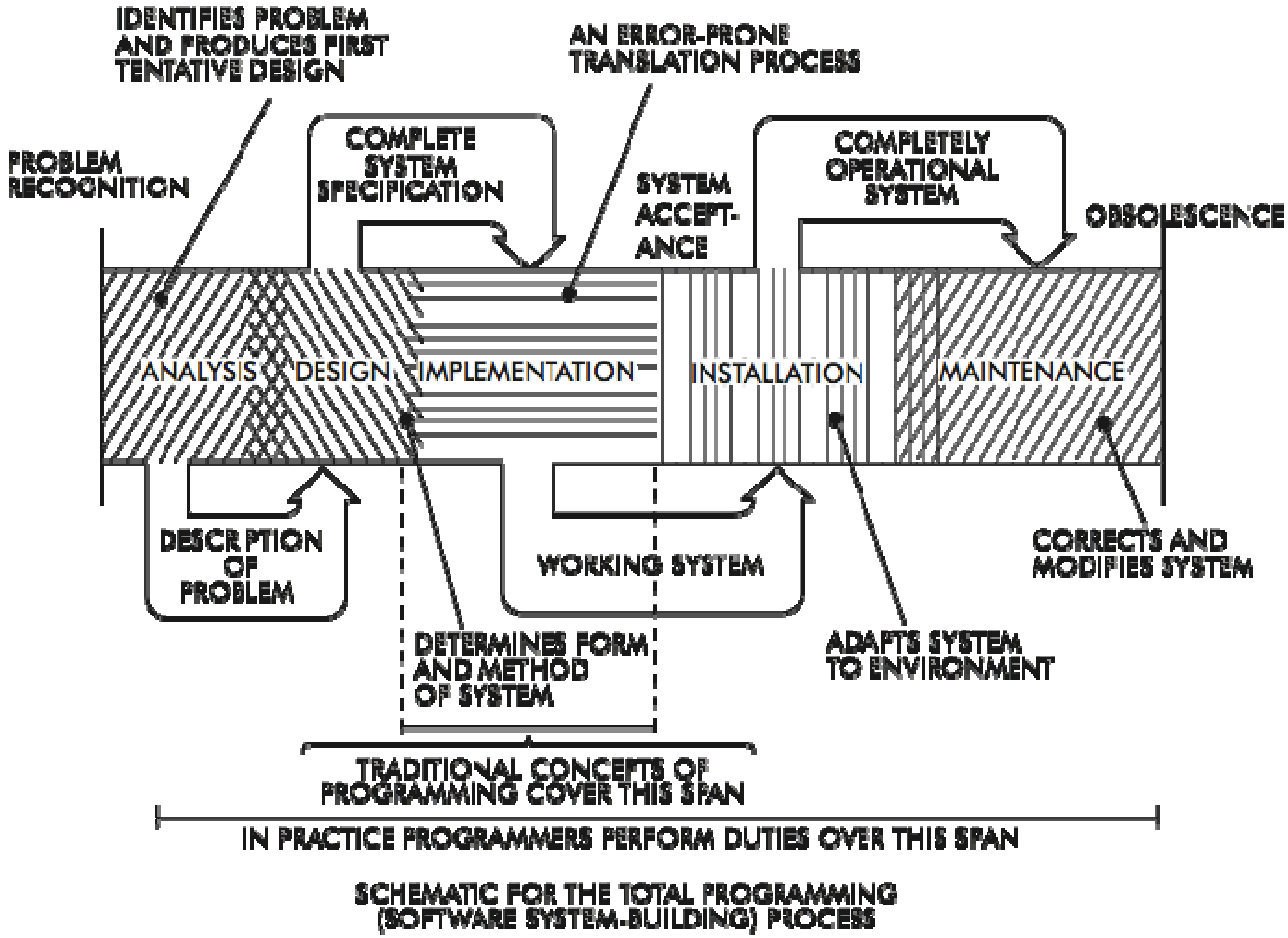


Figure 2. From Selig: Documentation for service and users. Originally due to Constantine.

SWEBOK - First Five KAs

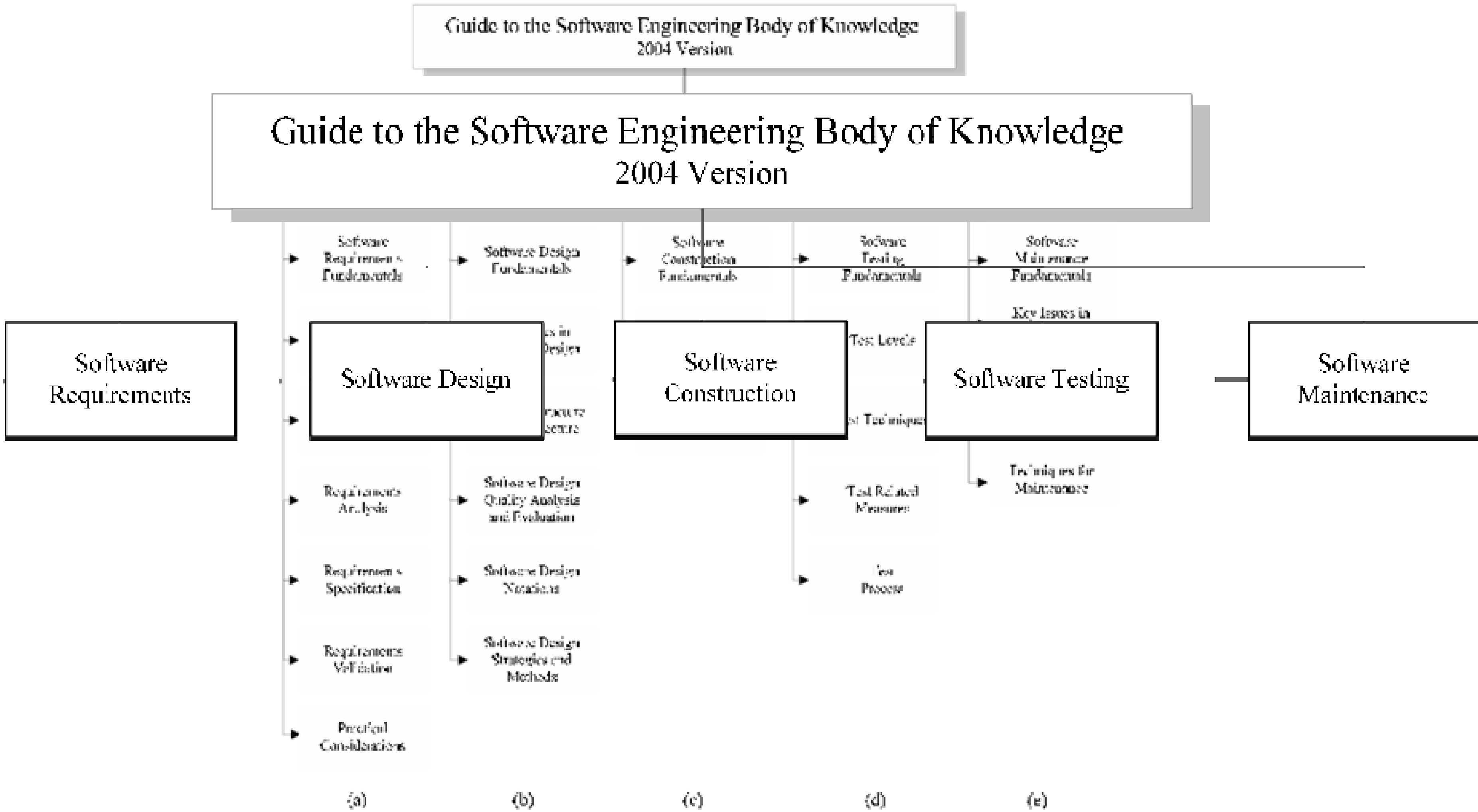


Figure 2 First five KAs

SWEBOK - Last Six KAs

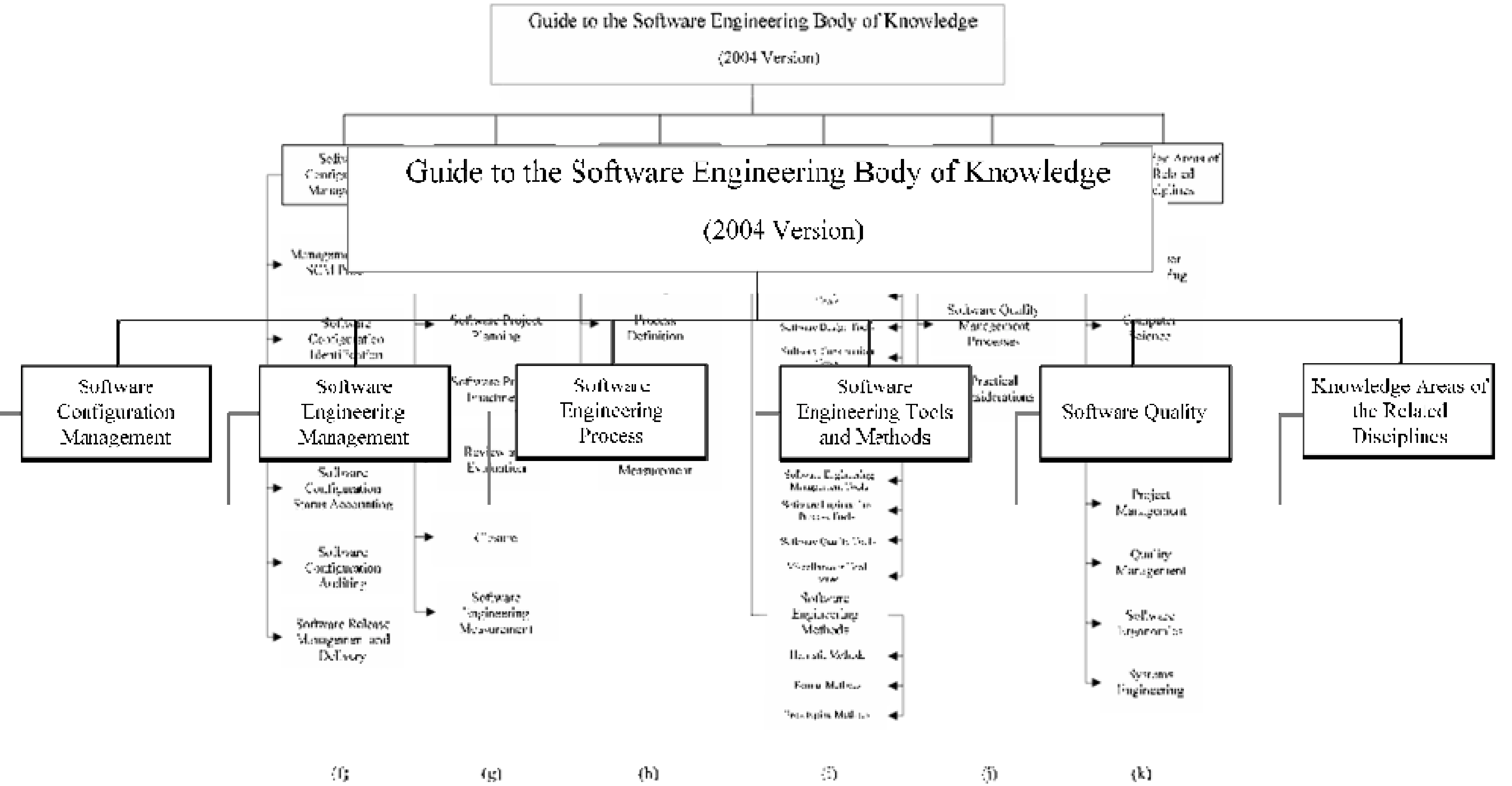


Figure 3 Last six KAs

Effort distribution in percentages

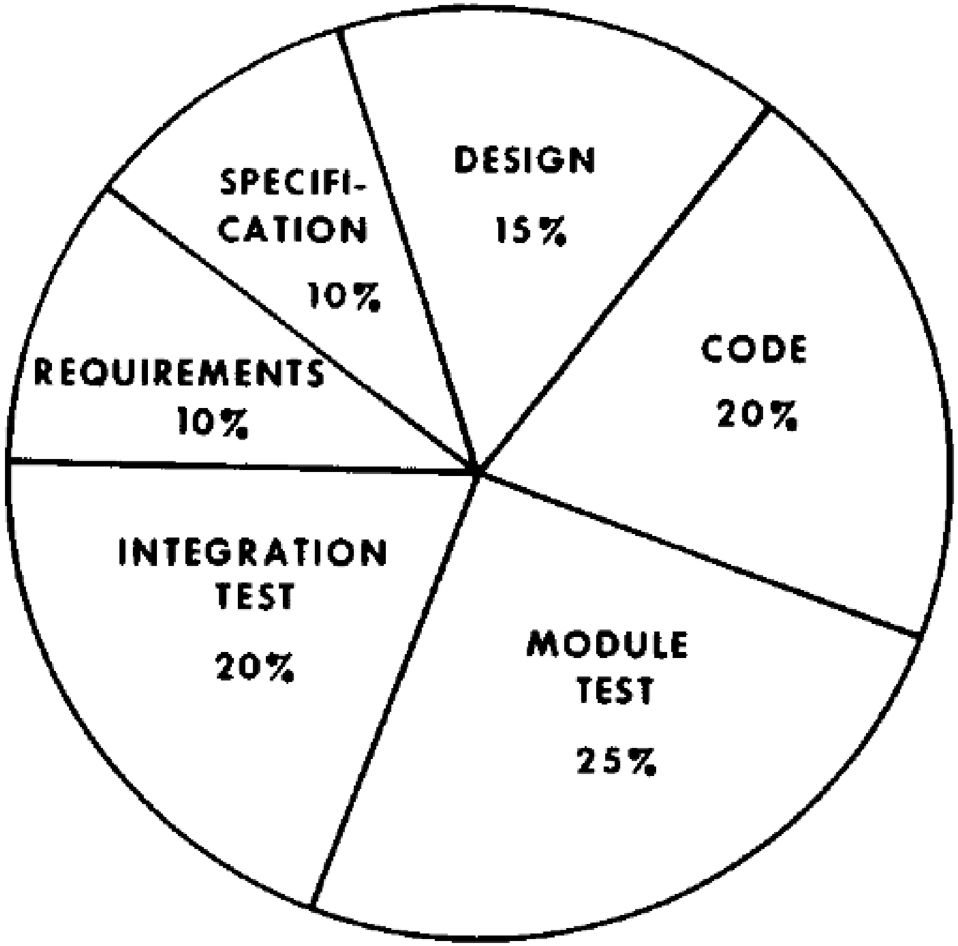


FIGURE 1. Effort required on various development activities (excluding maintenance)

Design, Structure Diagram

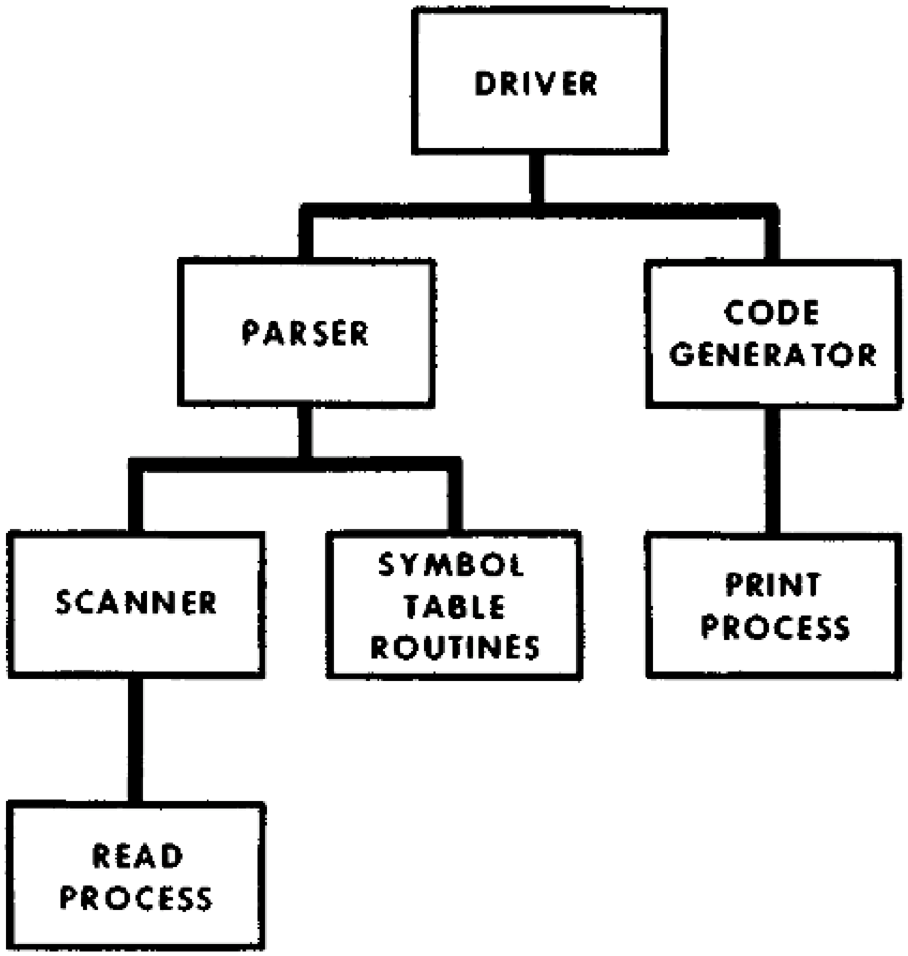
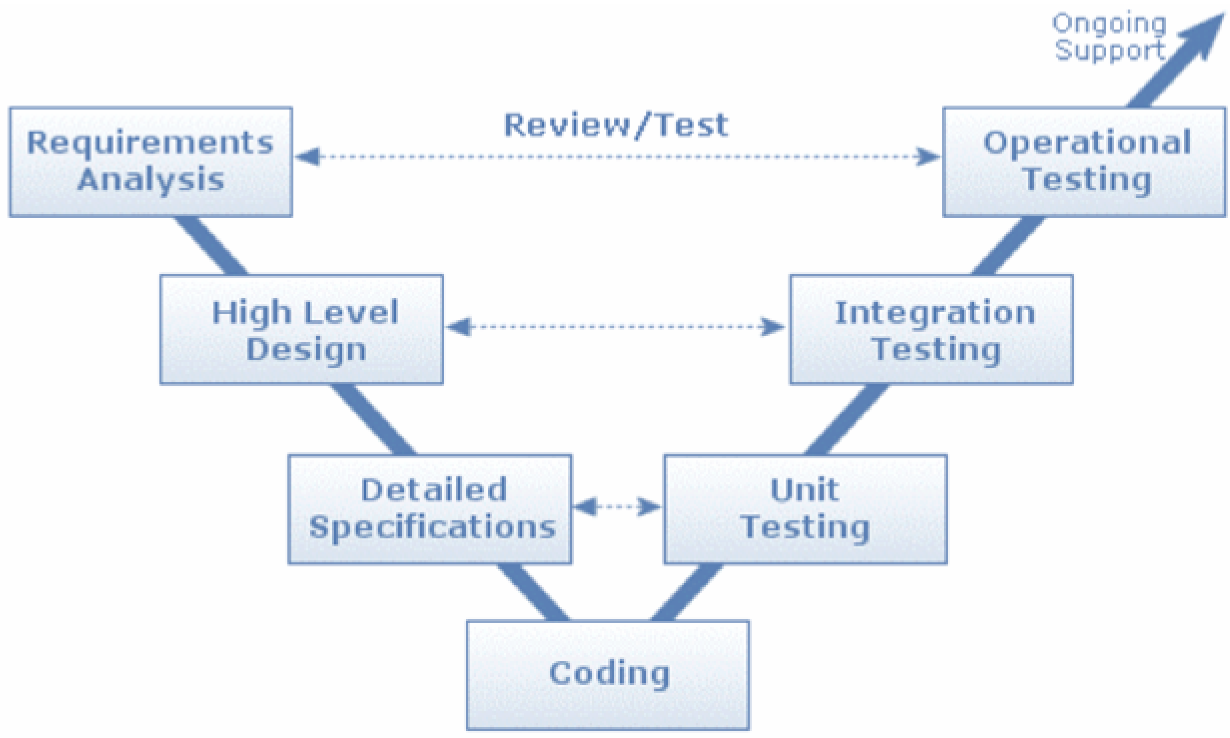


FIGURE 2. Sample baseline diagram for a compiler.

Testing

Unit test
Integration test
System test
Acceptance test



V-Model

Life-cycle Effort Distribution

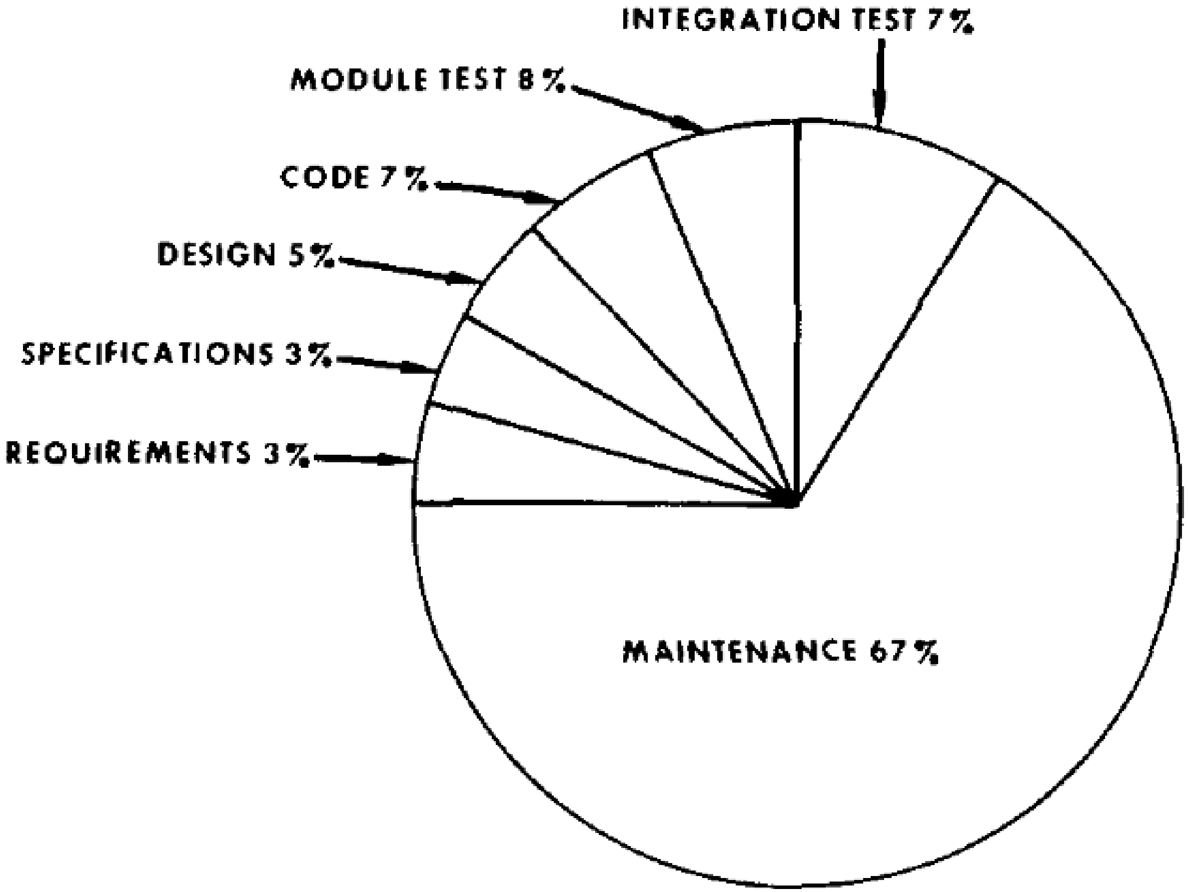


FIGURE 3. True effort on many large-scale software systems.

Cost of Communication

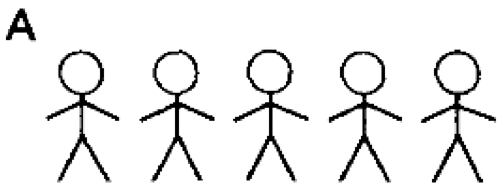


FIGURE 4(a) Single projects. 5,000 lines per year = 50,000 lines in two years (no communication between programmers)

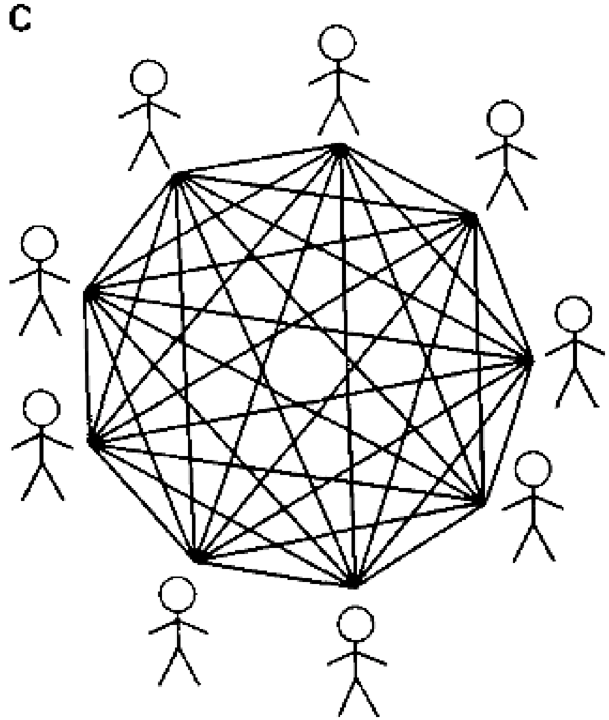


FIGURE 4(c). Nine-member team: 3,000 lines per year = 50,000 lines in two years (36 communication pairs).

Chief Programmer Team

B

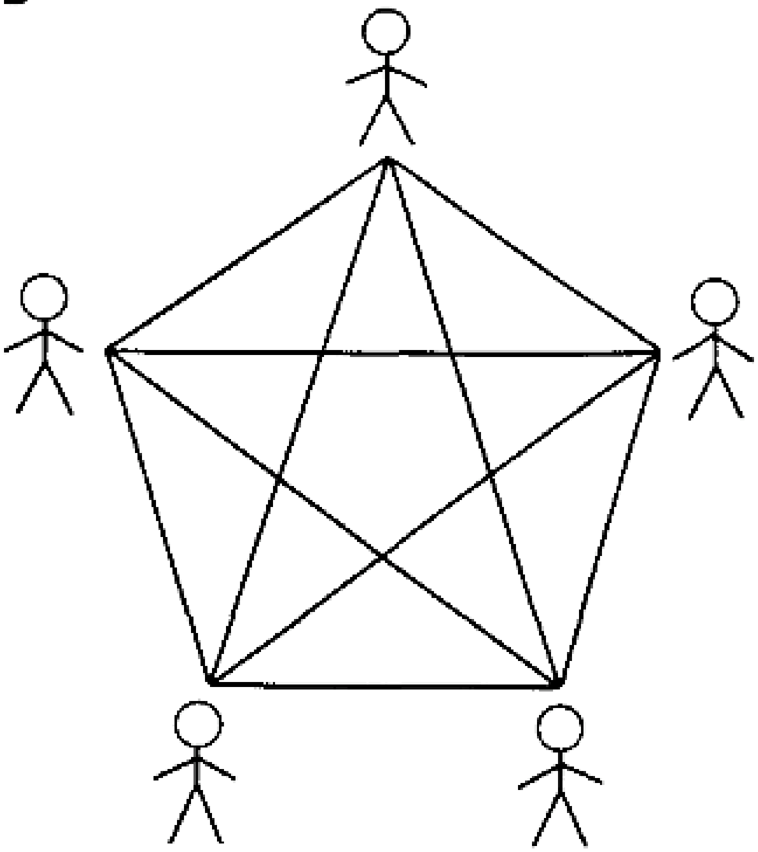


FIGURE 4(b). Five-member group: 4,000 lines per year = 40,000 lines in two years (ten communication pairs).

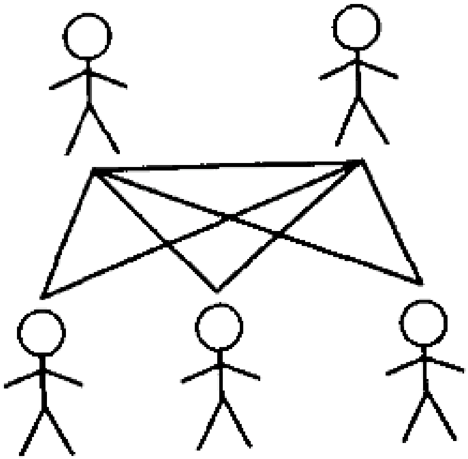


FIGURE 5. Fewer communications paths in a chief programmer team.

Estimations

Comparing to previous projects (analogy)
Decomposing the effort in smaller parts
Schedule work and estimate resources by the month (work breakdown structure)
Develop standards (basis for norms)

Rayleigh Curve

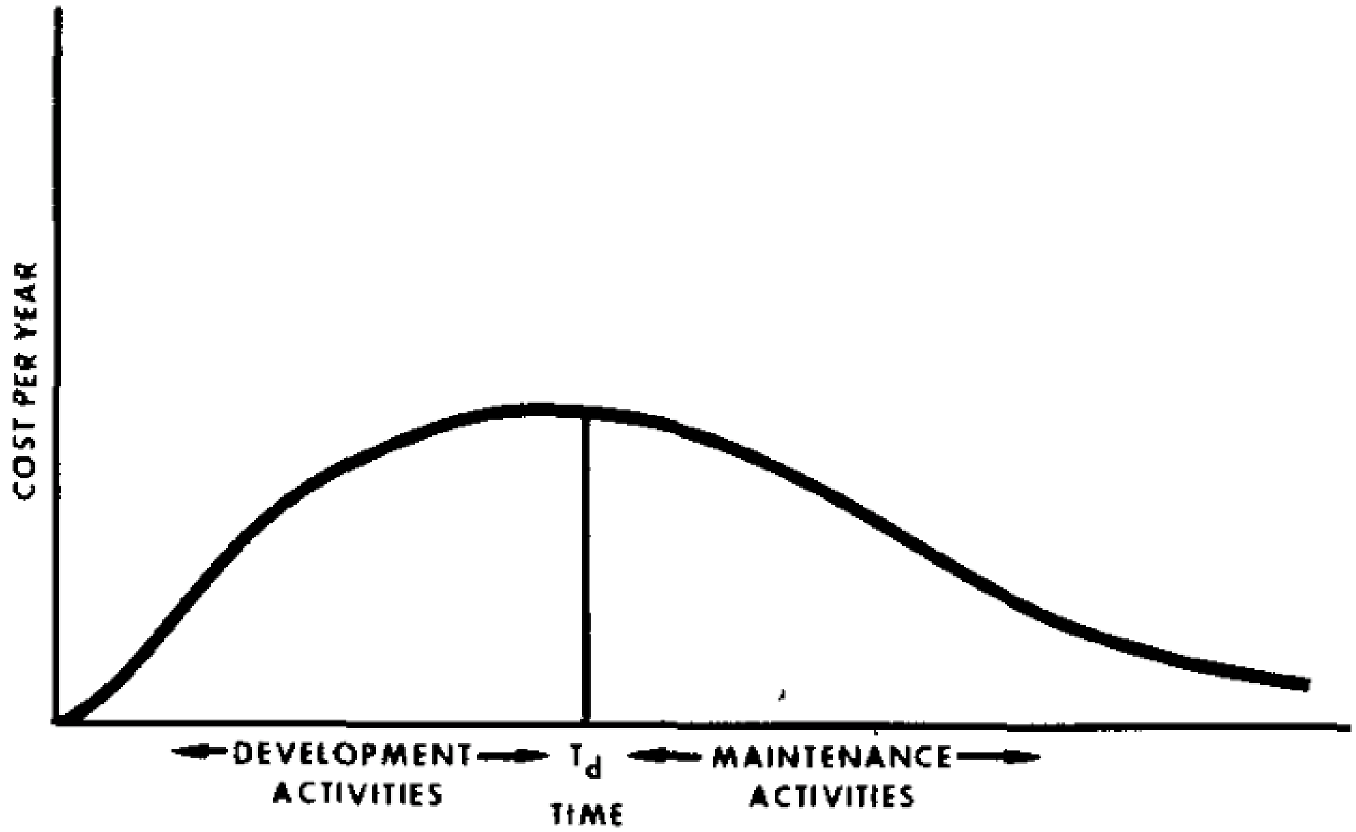


FIGURE 6. Yearly rate of expenditures approximates the Rayleigh curve. Total cost (area under curve) $= K$, $a = 1/T_d^2$, rate $= 2Kate^{-at^2}$

Nato-seminar in Garmisch 1968

NATO SOFTWARE ENGINEERING CONFERENCE 1968

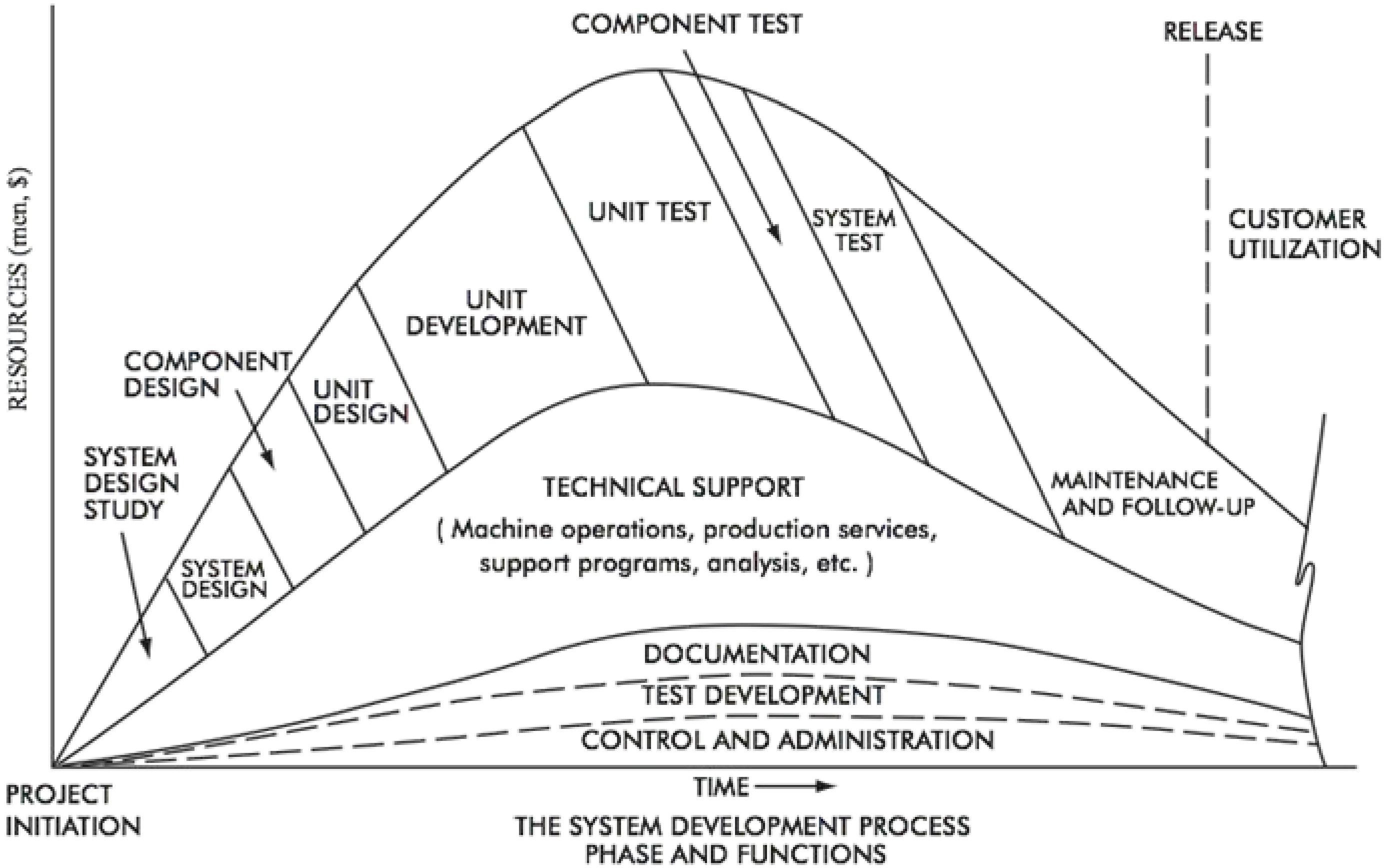
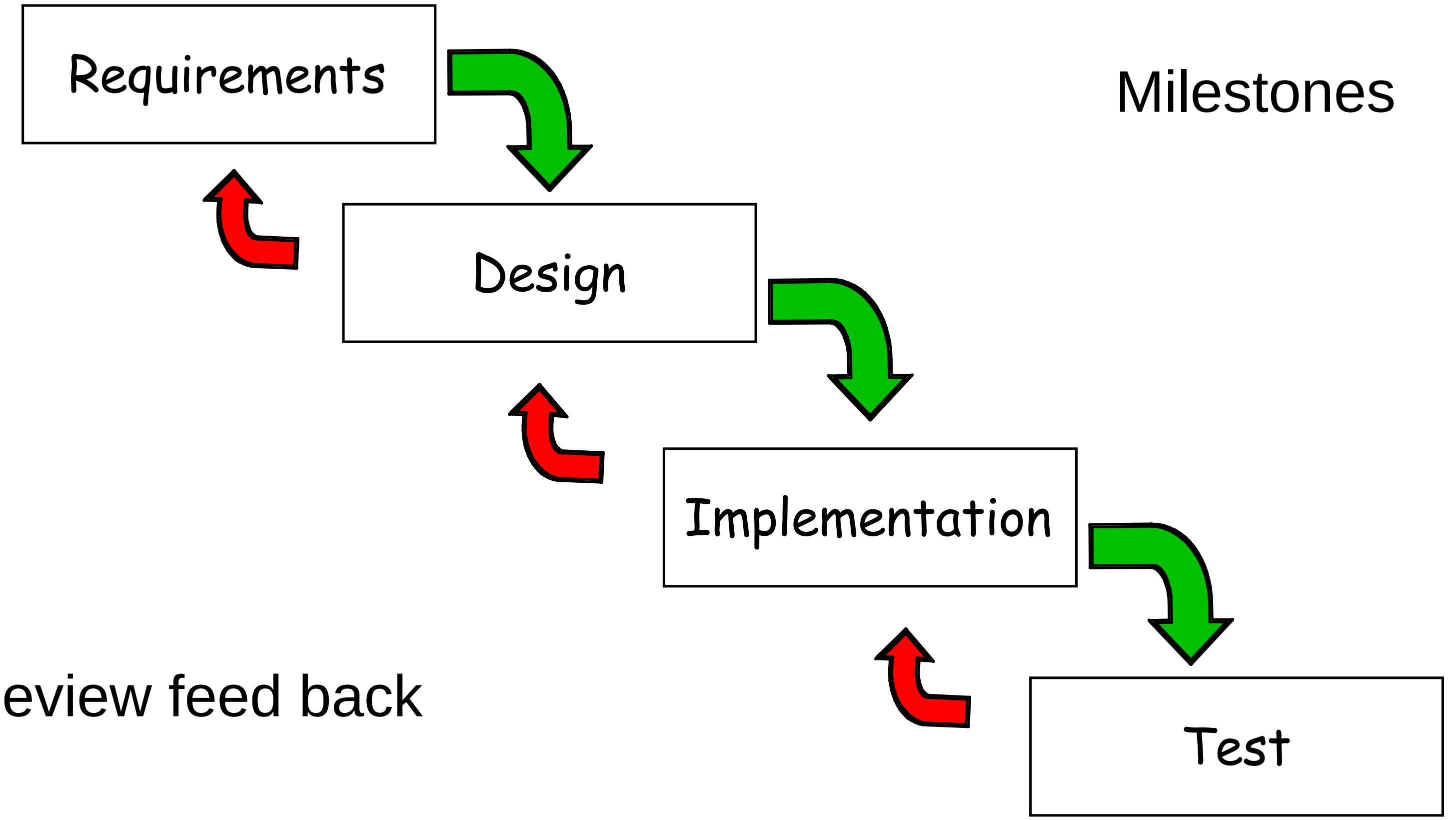


Figure 1. From Nash: Some problems in the production of large-scale software systems.

Waterfall Model



Milestones

Review feed back

Program Verification

FIGURE 7. Assertions A_i and A_j surround each statement of a program.

FIGURE 8. Predicates A_1 and A_n specify input-output behavior of a program.

Go-to statement considered harmful

Letters to the Editor

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

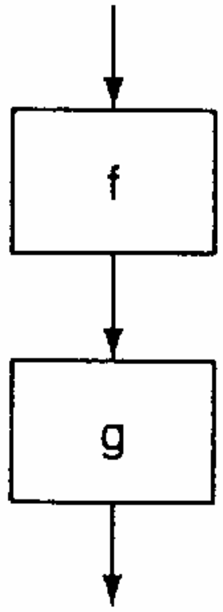
My first remark is that, although the programmer's activity

dynamic
call of t
we can c
textual
dynamic
Let us
or **repea**
superflu
recursiv
clude th
mented
the othe
makes u
processe
the repe
describe
a repeti
namic in
correspo

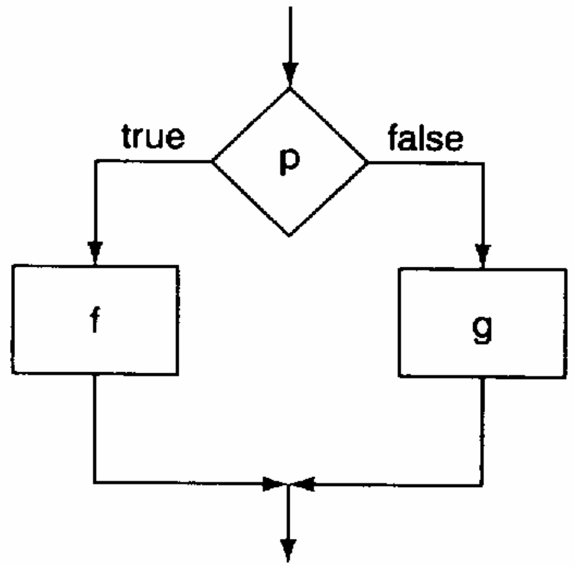
DIJKSTRA, E. "GOTO statement considered harmful," *Commun. ACM* 11, 3 (March 1968), 147-148.

Structured Programming

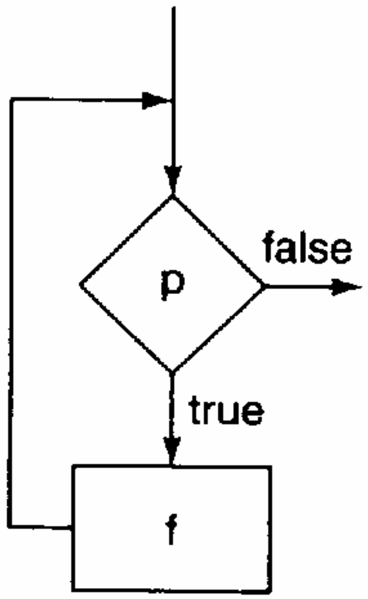
Sequence



Selection



Iteration



Minimize use of “goto” statements

Use sequence, selection, and iteration building blocks

Optimization I

$$A = B + C + D$$

(a) Temp1 = B + C
(b) Temp2 = Temp1 + D
(c) A = Temp2

1. Load register with B (from (a))
2. Add C to register
3. Store register in Temp1
4. Load register with Temp1 (from (b))
5. Add D to register
6. Store register in Temp2
7. Load register with Temp2 (from (c))
8. Store register in A

Optimization 2

A = B + C + D

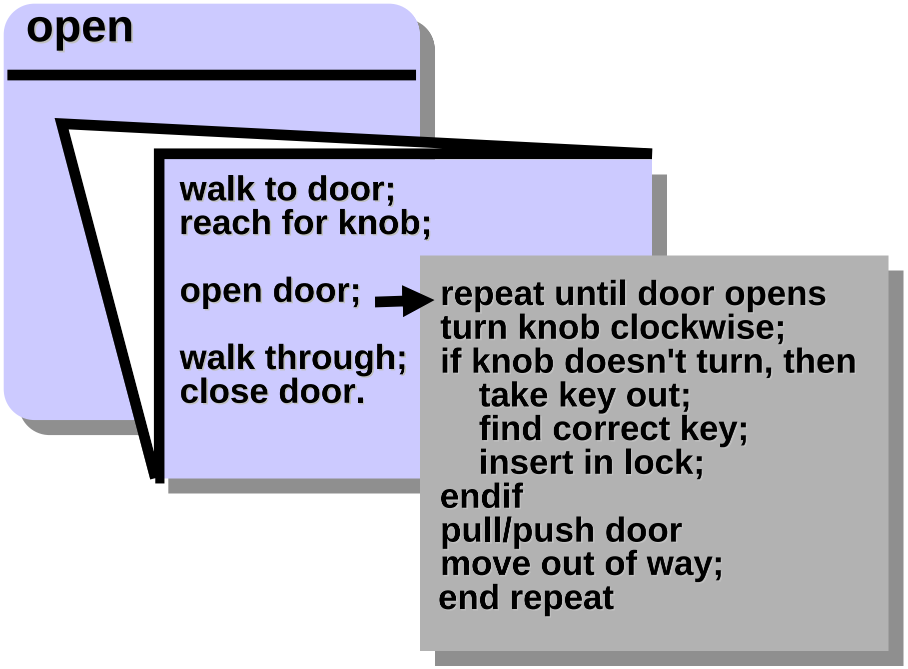
(a) Temp1 = B + C
(b) Temp2 = Temp1 + D
(c) A = Temp2

1. Load register with B (from (a))
2. Add C to register
3. Store register in Temp1
4. Load register with Temp1 (from (b))
5. Add D to register
6. Store register in Temp2
7. Load register with Temp2 (from (c))
8. Store register in A

Redundant Instructions

System Design

- Top Down Development
- Stubs
- Iterative Enhancement
- Throw away first version



Stepwise refinement

Boehm's Seven Principles

- Manage using a sequential life cycle plan
- Perform continuous validation
- Maintain disciplined product control
- Use enhanced top-down structured programming
- Maintain clear accountability
- Use better and fewer people
- Maintain commitment to improve process