Programming as *theory building*

Different views on programming by Parnas, Dijkstra and Naur.

The view of Peter Naur (1985)

Contents

1. Introduction
2. Programming and the Programmers Knowledge
3. Ryle’s notion of theory
4. The theory to be built by the programmer
5. Problems and costs of program modifications
6. Program life, death and revival
7. Method and theory building
8. Programmer’s status and the theory building view
9. Conclusions

1. Introduction

What is programming?

Common View

Programming
=
production of program texts
(incl. documentation)

Naur's View

Programming
=
activity by which the
programmers achieve a
certain kind of insight, a
theory, of the matter at hand

1. Introduction

What is programming?

Common View

Programming
=
production of program texts
(incl. documentation)

THE PRODUCTION VIEW

Naur's View

Programming
=
activity by which the
programmers achieve a
certain kind of insight, a
theory, of the matter at hand

THE THEORY BUILDING VIEW

1. Introduction

What is programming?

THE PRODUCTION VIEW

Common View

The notion of a programming method understood as a set of rules of the procedure to be followed is *based on invalid assumptions*

THE THEORY BUILDING VIEW

Naur's View

The notion of *program life* that depends on the continuous support of the program by programmers having its **theory**

2. Programming
& the programmers knowledge

Context

Case 1

Case 2

Conclusion

2. Programming
& the programmers knowledge

Context

Case 1

Case 2

Conclusion

The full program text and additional documentation is insufficient in conveying to even the highly motivated group B. The deeper insight into the design, that **theory** which is immediately present to the members of group A.

2. Programming & the programmers knowledge

Context

Case 1

Case 2

Conclusion

The full program text and additional documentation is insufficient in conveying to even the highly motivated group B. The deeper insight into the design, that **theory** which is immediately present to the members of group A.

The program text and its documentation has proved insufficient as a carrier of some of the most important design ideas.

2. Programming & the programmers knowledge

Context

Case 1

Case 2

Conclusion

A certain kind of knowledge possessed by a group of programmers.

The full program text and additional documentation is insufficient in conveying to even the highly motivated group B. The deeper insight into the design, that **theory** which is immediately present to the members of group A.

The program text and its documentation has proved insufficient as a carrier of some of the most important design ideas.

3. Ryle's notion of *theory*

Introduction

Programmer's knowledge should be regarded as a *theory* in the sense of Ryle (1949).

Description:

Merely Intelligent

Intelligent Behaviour

3. Ryle's notion of *theory*

Introduction

Programmer's knowledge should be regarded as a *theory* in the sense of Ryle (1949).

Description:

Merely Intelligent

Display particular knowledge of facts ('De slimste mens')

Intelligent Behaviour

3. Ryle's notion of *theory*

Introduction

Programmer's knowledge should be regarded as a *theory* in the sense of Ryle (1949).

Description:

Merely Intelligent

Display particular knowledge of facts ('De slimste mens')

Intelligent Behaviour

The ability to do certain things such as...

- to make and appreciate jokes
- to talk grammatically
- to fish

3. Ryle's notion of *theory*

Introduction

Programmer's knowledge should be regarded as a *theory* in the sense of Ryle (1949).

Description:

Merely Intelligent

Display particular knowledge of facts ('De slimste mens')

Intelligent Behaviour

The ability to apply criteria to detect & correct lapses, to learn from examples of others, etc.

3. Ryle's notion of *theory*

Introduction

Description:

Merely Intelligent

Intelligent Behaviour

Person following rules? 🤔

Person building & having a theory 😃

3. Ryle's notion of *theory*

Person following rules?

If the exercise of intelligence depended on following rules, there would have to be:

- rules about how to follow rules
- rules about how to follow rules about following rules
- ...

=> **Infinite Regress!**

3. Ryle's notion of *theory*

Introduction

Description:

Merely Intelligent

Intelligent Behaviour

Person following rules?

If the exercise of intelligence depended on following rules, there would have to be:

- rules about how to follow rules
- rules about how to follow rules about following rules
- ...

🤔

=> **Infinite Regress!**

Person building & having a theory 😃

4. The theory to be built by the programmer

The programmer's knowledge transcends that given in documentation in at least three areas:

1. The programmer can explain for each part of the program text what activity of the world is matched by it.

2. The programmer can explain *why* each part of the program text is what it is.

3. The programmer is able to respond constructively to any demand for program modification.

5. Problems and cost of program modifications

1. Program modifications
2. Flexibility
3. Why the *Theory Building View* makes sense.

5. Problems and cost of program modifications

1. Program modifications

   One hopes to achieve a saving of costs by making modifications of an existing program text.

   It is assumed that the dominating cost is one of *text manipulation.*

   Contradicts with Naur's view

2. Flexibility

3. Why the *Theory Building View* makes sense.

5. Problems and cost of program modifications

1. Program modifications
2. Flexibility

   It is often stated that programs should be designed to include a lot of *flexibility* so as to be readily adaptable to changing circumstances.

   E.g. Object Oriented Programming

   However, flexibility can in general only be achieved at a substantial cost

3. Why the *Theory Building View* makes sense.

5. Problems and cost of program modifications

1. Program modifications
2. Flexibility

Why the *Theory Building View* makes sense.

The kind of similarity that has to be recognized is accessible to the human beings who possess the *theory* of the program

although

entirely outside of the reach of what can be determined by rules, since even the criteria to judge it cannot be formulated.

On the basis of the *Theory Building View* the decay of a program text as a result of modifications made by programmers without a proper grasp of the underlying *theory* becomes understandable.

6. Program *Life*, *Death* and *Revival*

1. Program life
2. Program death
3. Program revival
4. Extended life of a program
5. Education of new programmers
6. Two main messages

6. Program *Life*, *Death* and *Revival*

During **program life** a programmer team possessing its *theory* remains in active control of the program and in particular retains control over all modifications.

6. Program *Life*, *Death* and *Revival*

The **death of a program** happens when the programmer team possessing its *theory* is dissolved.

6. Program *Life*, *Death* and *Revival*

The **revival of a program** is the rebuilding of its *theory* by

a new programmer team.

6. Program *Life*, *Death* and *Revival*

The **extended life of a program** depends on the taking over by new generations of programmers of the *theory* of the program.

- ➔ *Insufficient* to become familiar with the program text & other documentation.
- ➔ *Necessary* that new programmers work closely with programmers who already possess the *theory*

6. Program *Life*, *Death* and *Revival*

The **education of new programmers** is similar to

learning to play a *musical instrument*.

6. Program *Life*, *Death* and *Revival*

Two messages:

1. **Program Revival** is strictly impossible
2. **Preferably**, the *program text* should be **discarded** & start from scratch building a (new) *theory* with the new team.

7. Method and theory building

=

Clarify the relation between the *Theory Building View* & the notions behind programming methods

8. Programmer's status on the *Theory Building View*

Much of the current discussions seem to assume that

programming is similar to *industrial production*:

- programmers controlled by rules
- programmers can easily be replaced

Computer Science &
Descriptions

versus

Software Engineering &
Prescriptions

Contents

1. Is computer science really a science?

... anything that calls itself a 'science' probably isn't ...

-- Searle, 1984

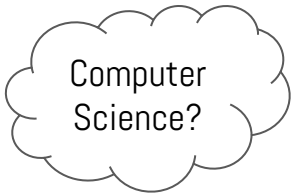
2. Two mechanisms

Scientific Mechnanism

Engineering Mechanism

2. Two mechanisms

Scientific Mechnanism — choose a *model* that is faithful to the *target*



Engineering Mechanism — produce a *target* that is faithful to the *model*

3. Descriptions and Prescriptions

Scientific Mechnanism — choose a *model* that is faithful to the *target*

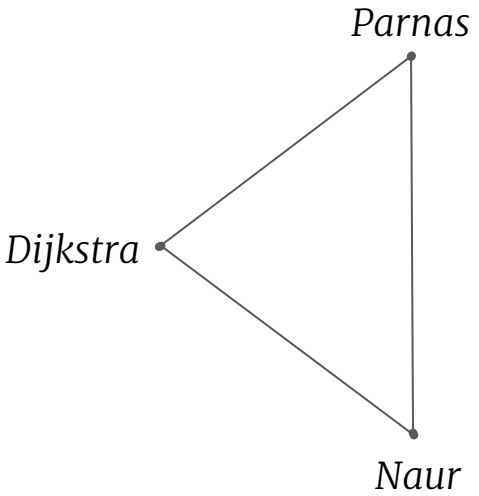
= choosing a good ***description***, which is an account of some event

Engineering Mechanism — produce a *target* that is faithful to the *model*

= abiding by a ***prescription***, which is a recommendation that is authoritatively put forward

4. Different views to a fundamental problem

Software development by Parnas, Dijkstra and Naur.



“We will never find a process that allows us to design software in a perfectly rational way.”

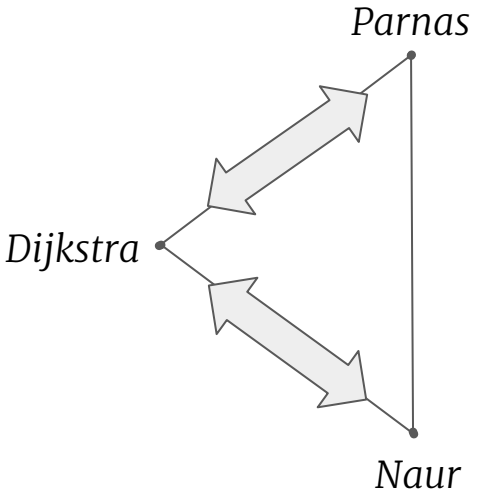
...

“The good news is that we can ‘fake’ the actual irrational design process. We can represent our system to others ***as if we had been rational designers*** *and it pays to pretend to do so during development and maintenance.”*

“The notion of the programmer as an easily replaceable component in the program production activity has to be abandoned.”

4. Different views to a fundamental problem

Software development by Parnas, Dijkstra and Naur.



“We will never find a process that allows us to design software in a perfectly rational way.”

...

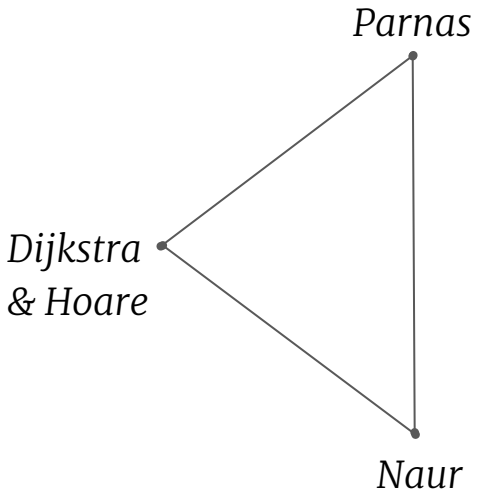
*“The good news is that we can ‘fake’ the actual irrational design process. We can represent our system to others **as if we had been rational designers** and it pays to pretend to do so during development and maintenance.”*

“The notion of the programmer as an easily replaceable component in the program production activity has to be abandoned.”

5. Overview

Software development by Parnas, Dijkstra and Naur.

*We can get a rational & complete **prescription** on how to (rationally) develop 'correct-by-construction' software.*



*We can get an accurate **description** of the software design process.*

*We **cannot** even get an accurate **description** of the software design process.*