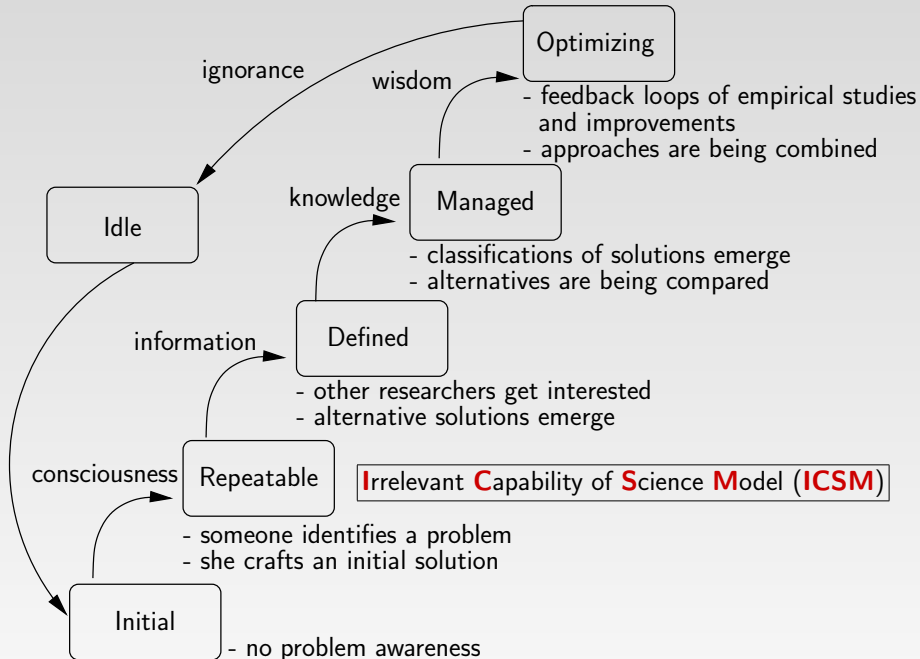
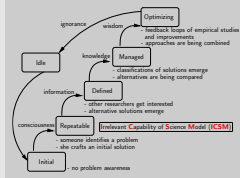


# Panel: Identifications of Concepts, Features, and Concerns in Source Code





## └ Raise and Fall of Research Areas



The transition from "optimizing" to "idle":

- progress comes at increasingly higher prices (all low-hanging fruits are eaten up)
- researchers loose interests
- researchers find that this problem is unsolvable in the first place
- researchers are in grief and despair: they cannot see any impact of their work on the work of practitioner
- practitioner note that all that has never been a problem in the first place


In fact, the stages are not a stairway to heaven, but a circle of life. There comes a stage after "optimizing" when wise people get used to be gurus. Gurus tend to loose contact to the ground when floating one meter above the ground.



# Where are we today?

indicators for stage "defined":

- 2002: Norman Wilde receives "Most influential paper award of ICSM 1992"
- several approaches exist
- first empirical comparison (Wilde et al., 2001)
- ongoing topic:
  - paper session at VISSOFT
  - paper session at ICSM
  - feature-location paper received best paper award at ICSM'05 and ICSM'
  - this panel
  - paper session at SCAM



# How do we get to the next level?

- more empirical studies
  - to compare the approaches
  - to study what the problems of programmers are in detail; how programmers handle this problem today; how the approaches would in their daily life
- rethinking some of the fundamental assumptions



# Rethinking some of my own assumptions

- "use" does not equal "execute"
- isolated features versus feature combination (Koschke and Quante, 2005)
- specific versus general features
- feature location versus testing

└ Problems of dynamic analyses



└ Rethinking some of my own assumptions

- "use" does not equal "execute"
- isolated features versus feature combination (Koschke and Quante, 2005)
- speci c versus general features
- feature location versus testing

- "use" does not equal "execute"
- isolated features versus feature combination

{ "traditional" difference-based dynamic feature location tends to focus on isolated features; concept analysis generalized these approaches to multiple features:

{ our case study: the sum is more than its parts


{ combinatorial explosion of scenarios/test cases for combined features (e.g., Gnu C compiler cc1 has more than 500 options; its input, the language, has manifold combinations of language features)

- speci c versus general

{ application speci c features may be found with dynamic difference-based features

{ how do we detect general infrastructure features; aspects (or cross-cutting concerns) are features that are spread all over the code; they are often executed for any scenario: how can we detect that? dynamic difference-based analysis?

└ Problems of dynamic analyses

└  Rethinking some of my own assumptions

- "use" does not equal "execute"
- isolated features versus feature combination (Koschke and Quant, 2005)
- specific versus general features
- feature location versus testing

- feature location versus testing

- { test cases are not really suited for dynamic difference-based feature location

- { hence, we need to design new test cases

- { white box testers can leverage knowledge of implementation

- { which knowledge can we assume for feature location?

Koschke, R. and J. Quante (2005, November). On dynamic feature location. In Proceedings of the International Conference on Automated Software Engineering.

Wilde, N., M. Buckellew, H. Page, and V. Rajlich (2001, March). A Case Study of Feature Location in Unstructured Legacy Fortran Code. In Proceedings of the 5th European Conference on Software Maintenance and Reengineering, Lisbon, Portugal, pp. 68{75. IEEE Computer Society Press.