Linus Ververs
AG Software Engineering
Institut für Informatik
Fachbereich Mathematik & Informatik

Freie Universität Berlin

Qualitative Forschung zu Paar-Programmierung

1. Paar Programmierung
2. Grounded Theory Methodology
3. Forschung der AGSE
4. Ergebnisse + Beispiele aus den letzten 8 Monaten
5. Forschungsprozess & Ausblick

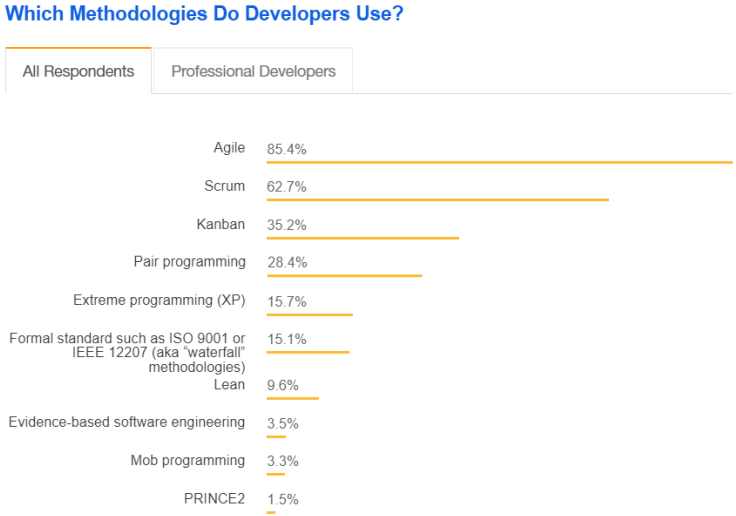
Warum forschen wir?

> To but it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, **programming has become an equally gigantic problem**. – Edsger W. Dijkstra 1972 [1]

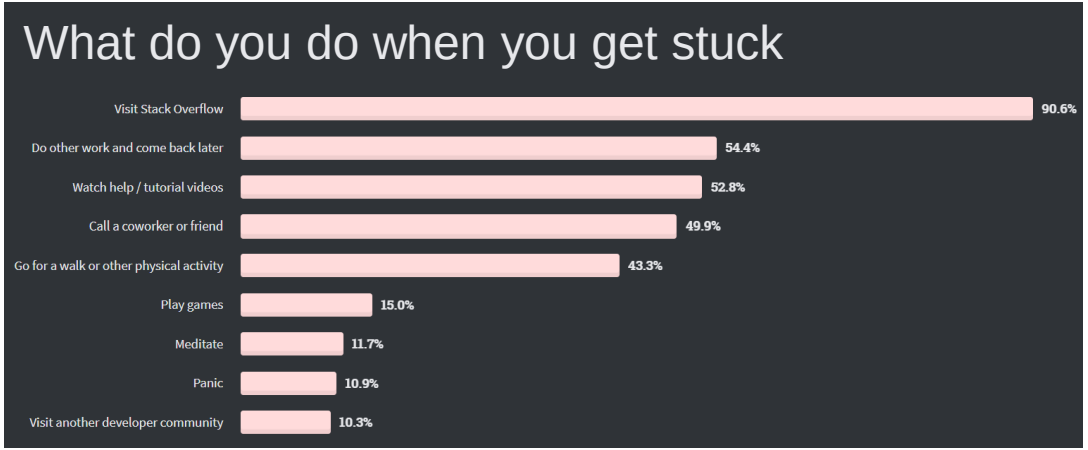
Ziel der Software Engineering Forschung: Evidenzbasierte Empfehlungen zur Verbesserungen des Softwareentwicklungsprozesses in der Praxis.

Was ist Paar-Programmierung?

“Pair programming (PP) is the practice of two developers working closely together on one computer to solve a technical task.” – Franz Zieris [2]



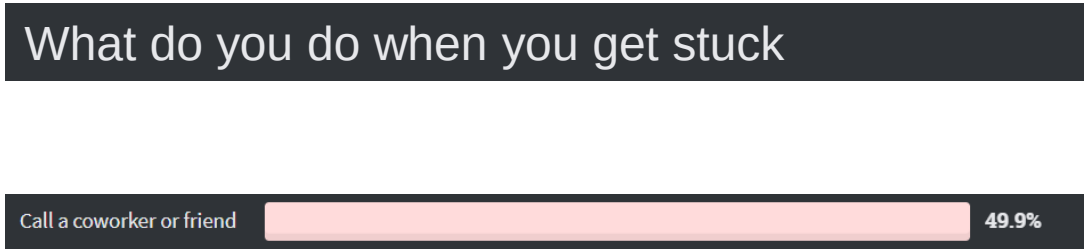
Stackoverflow.com-Umfrage 2018 [3]; n=58,981



Stackoverflow.com-Umfrage 2020 [4]; n≈65.000

Was ist Paar-Programmierung?

“Pair programming (PP) is the practice of two developers working closely together on one computer to solve a technical task.” – Franz Zieris [2]

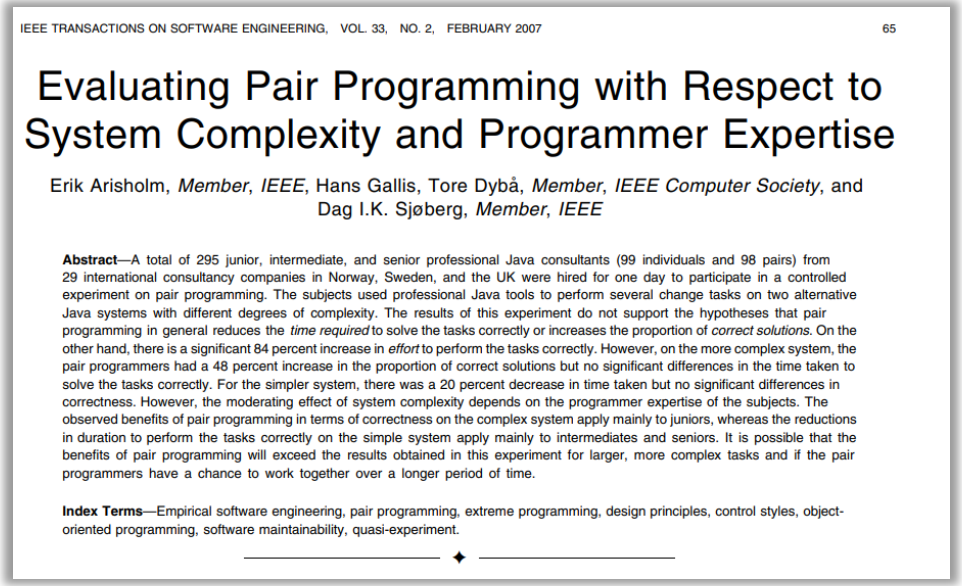


→ Vermutung: Das führt in vielen Fällen auch zu Pair-Programming

Warum diese Dissonanz???
→ PP als Practice vs. PP als Workmode
→ „Ach, das ist schon Pair Programming?!“

Forschung zu Paar-Programmierung

- Experimentelle Studie mit 295 Consultants
- Einteilung in Junior, Intermediate & Senior
- 99 Solo-Entwickler vs. 98 Paare
- Bearbeitung von 2 unterschiedlich komplexen Aufgaben



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 2, FEBRUARY 2007 65

Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise

Erik Arisholm, *Member, IEEE*, Hans Gallis, Tore Dybå, *Member, IEEE Computer Society*, and Dag I.K. Sjøberg, *Member, IEEE*

Abstract—A total of 295 junior, intermediate, and senior professional Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden, and the UK were hired for one day to participate in a controlled experiment on pair programming. The subjects used professional Java tools to perform several change tasks on two alternative Java systems with different degrees of complexity. The results of this experiment do not support the hypotheses that pair programming in general reduces the *time required* to solve the tasks correctly or increases the proportion of *correct solutions*. On the other hand, there is a significant 84 percent increase in *effort* to perform the tasks correctly. However, on the more complex system, the pair programmers had a 48 percent increase in the proportion of correct solutions but no significant differences in the time taken to solve the tasks correctly. For the simpler system, there was a 20 percent decrease in time taken but no significant differences in correctness. However, the moderating effect of system complexity depends on the programmer expertise of the subjects. The observed benefits of pair programming in terms of correctness on the complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors. It is possible that the benefits of pair programming will exceed the results obtained in this experiment for larger, more complex tasks and if the pair programmers have a chance to work together over a longer period of time.

Index Terms—Empirical software engineering, pair programming, extreme programming, design principles, control styles, object-oriented programming, software maintainability, quasi-experiment.

„However, on the **more complex system**, the **pair programmers had a 48 percent increase** in the proportion of **correct solutions** but no significant differences in the time taken to solve the tasks correctly.”[5]

Forschung zu Paar-Programmierung

- Experimentelle Studie mit 295 Consultants
- Einteilung in Junior, Intermediate & Senior
- 99 Solo-Entwickler vs. 98 Pa
- Bearbeitung von 2 unterschied
  Auf


IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 2, FEBRUARY 2007 65

Evaluating Pair Pro to System Com se

Erik Ari

Society, and

consultants (99 individuals and 98 pairs) from K were hired for one day to participate in a controlled al Java tools to perform several change tasks on two alternative he results of this experiment do not support the hypotheses that pair ed to solve the tasks correctly or increases the proportion of *correct solutions*. On the ent increase in *effort* to perform the tasks correctly. However, on the more complex system, the rcent increase in the proportion of correct solutions but no significant differences in the time taken to ty. For the simpler system, there was a 20 percent decrease in time taken but no significant differences in However, the moderating effect of system complexity depends on the programmer expertise of the subjects. The ed benefits of pair programming in terms of correctness on the complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors. It is possible that the benefits of pair programming will exceed the results obtained in this experiment for larger, more complex tasks and if the pair programmers have a chance to work together over a longer period of time.

Index Terms—Empirical software engineering, pair programming, extreme programming, design principles, control styles, object-oriented programming, software maintainability, quasi-experiment.

Experimente im Software Engineering:
Hohe **Glaubwürdigkeit** auf **Kosten der Relevanz**!

„H **more complex system**, the **pair programmers had a 48 percent inc se** in the proportion of **correct solutions** but no significant differences in the time taken to solve the tasks correctly.”[5]

Qualitative Feldforschung der AGSE

Betrachtung / Untersuchung der Softwareentwicklung in der Praxis.

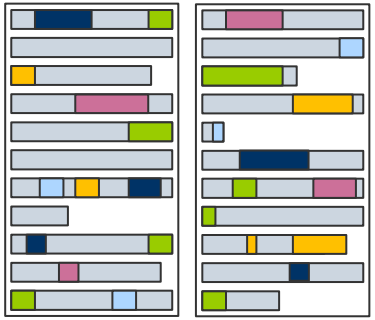
Qualitative, explorative Forschung um positive / negative Phänomene zu identifizieren & zu erklären.

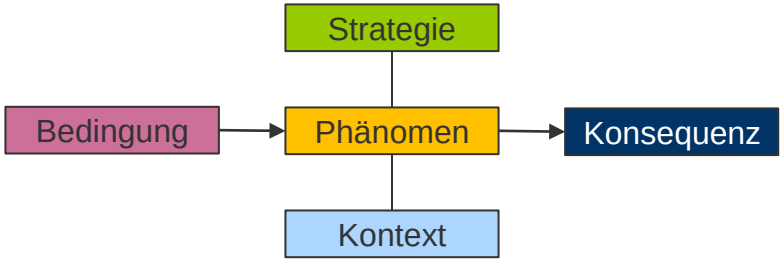
Formulieren von Theoriegebilden & Ableiten von Empfehlungen für die Softwareentwicklung in der Praxis.

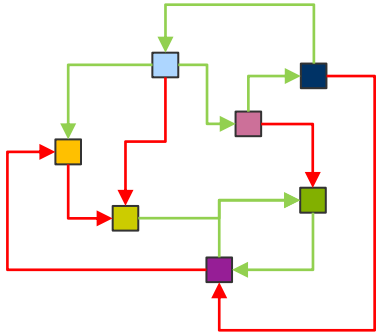
Verwendung der Grounded Theory Methodology aus den Sozialwissenschaften zur Bildung fundierter Theorien.

Grounded Theory Methodology*

Offenes Codieren → Axiales Codieren → Selektives Codieren







Theoretische Stichproben

Theoretische Sensitivität

Theoretische Sättigung

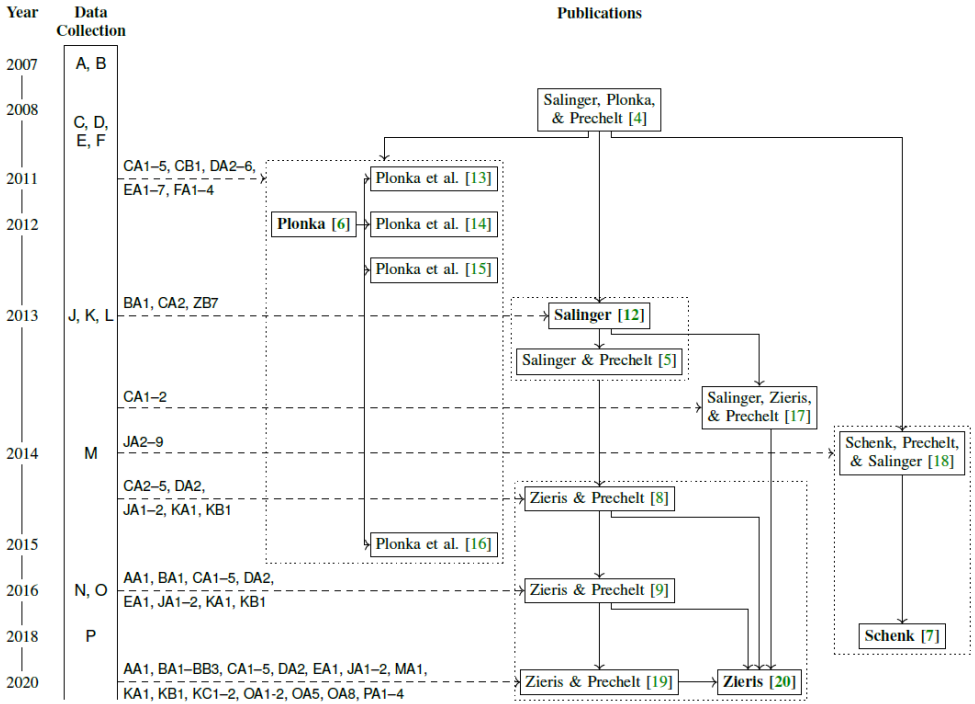
Theoretisches Codieren

Ständiges Vergleichen

*Hier sehr vereinfacht und verkürzt dargestellt.

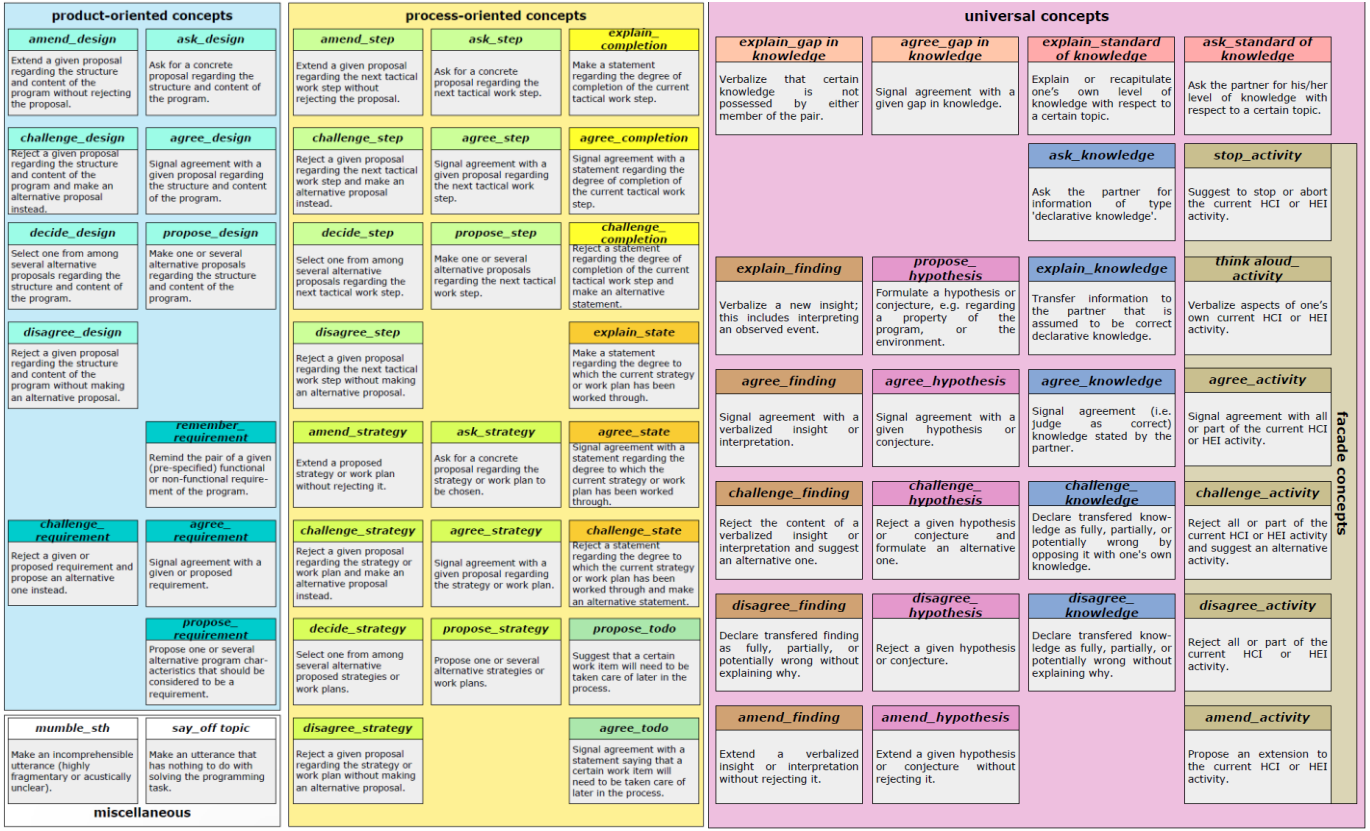
PPind: Datensatz der Arbeitsgruppe [6]

- Sammlung von PP-Sitzungen aus der Praxis seit 2007
- Insgesamt 67 Sitzungen mit ca. 100 Stunden Videomaterial
- Aus 13 Firmen mit 57 Entwicklern
- Diverse Technologien, Zielsetzungen und Paar-Konstellationen
- Klassische (vor-Ort) PP & Distributed (virtuelle) PP



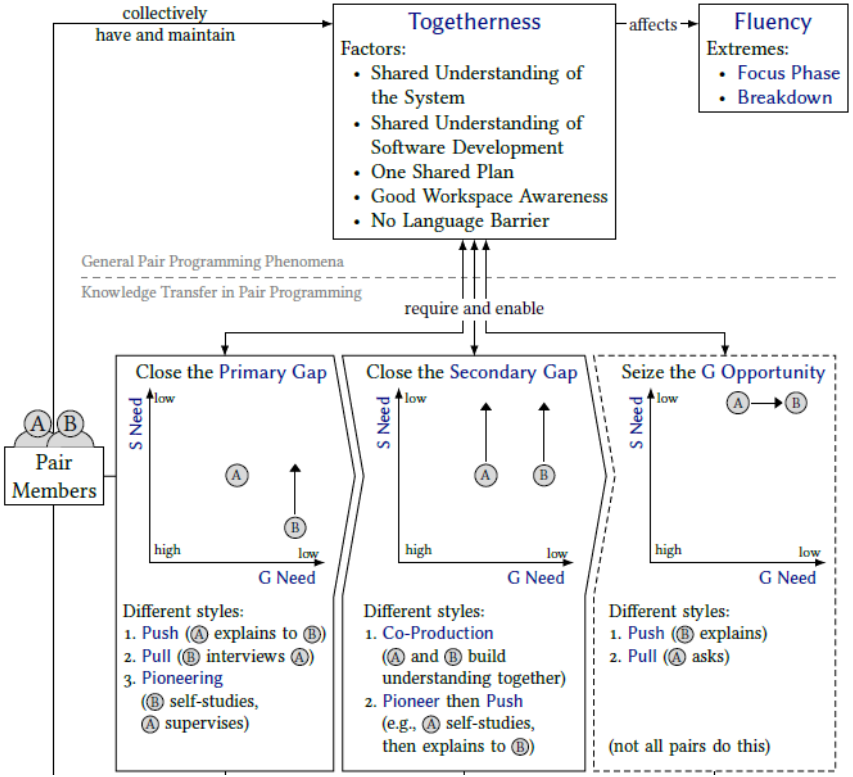
Grafik aus [6]

Base Layer für PP-Forschung (S. Salinger) [7]



Wissenstransfer in der PP (F. Zieris) [2]*

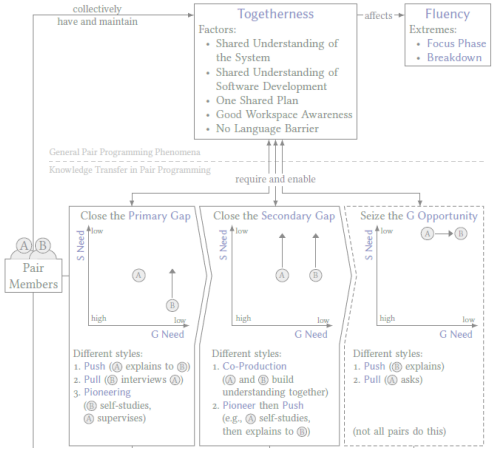
- Togetherness als fundamentales Konzept in der PP.
- Hohe Togetherness führt zu Focus Phases, niedrige zu Breakdowns
- Unterscheidung von S(pecific)- & G(eneral)-Knowledge
- Wissenstransfer und –erwerb in unterschiedlichen Modi



Grafik aus [2] *364 seitige Dissertation: Ergebnisse in Auszügen und verkürzt

Wissenstransfer in der PP (F. Zieris) [2]*

- Togetherness als fundamentales Konzept in der PP.
- Hohe Togetherness führt zu Focus Phases, niedrige zu Breakdowns
- Unterscheidung von S(pecific)- & G(eneral)-Knowledge
- Wissenstransfer und –erwerb in unterschiedlichen Modi

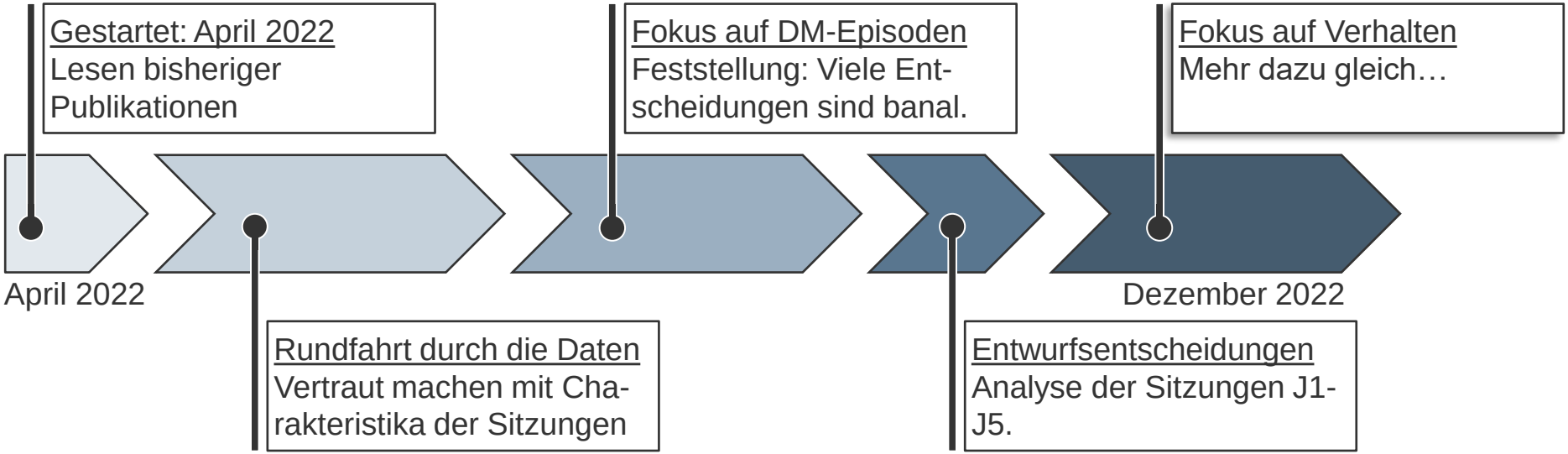


Grafik aus [2] *364 seitige Dissertation: Ergebnisse in Auszügen und verkürzt

Wie habe ich gearbeitet?

Forschungsprozess der letzten Monate & Beispielhafte Datenanalyse

Entscheidungsfindung in der PP (that‘s me)



Codierungsbeispiele

Beispiel: CA1

Observer: C1
Hat vor der Sitzung an der Aufgabe gearbeitet.

Driver: C2
Seit Beginn Driver und bleibt Driver für ca. 90% der Sitzung.

Neuartiges GUI-Element hinzufügen. Design und Verhalten ähnelt einer Komponente, ist aber nicht identisch.

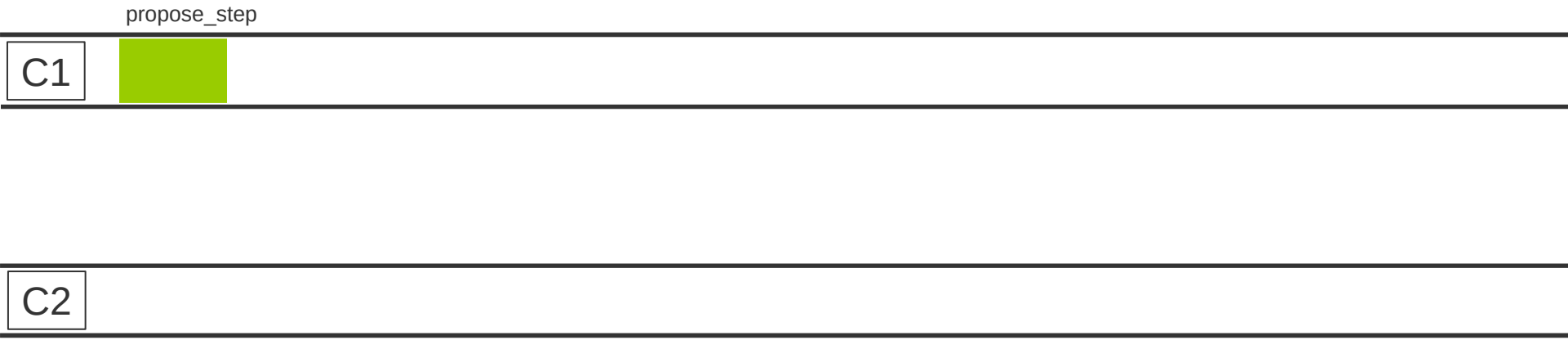
Der Clip beginnt mit dem Vorschlag von C1 eine Zeile, die er vor der Sitzung geschrieben hat, einzukommentieren und sich die Effekte im GUI anzuschauen.

Beispiel: CA1 0:15:20 - 0:17:12

C1

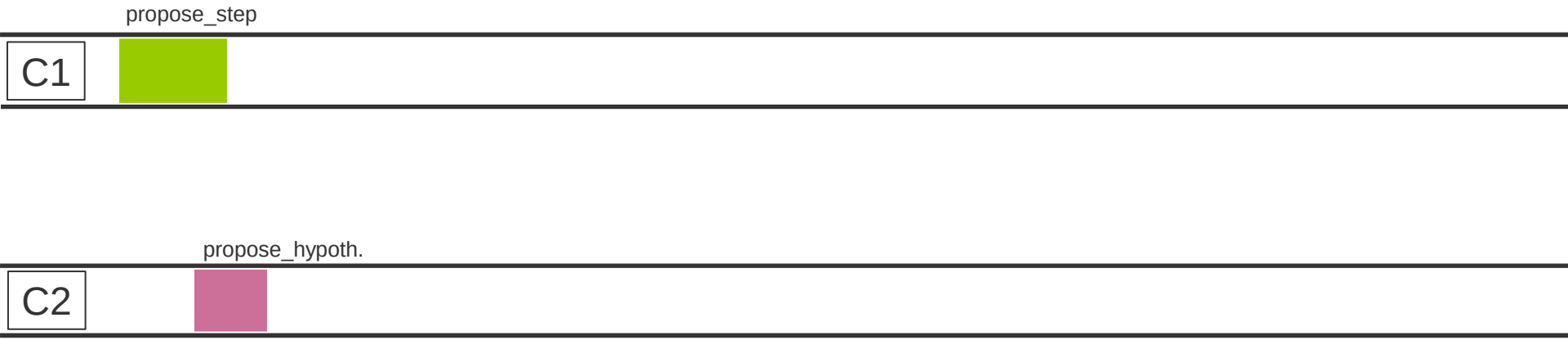
C2

Beispiel: CA1 0:15:20 - 0:17:12



„Du kannst ja mal einkommentieren und dann schauen wir uns in der Demo an, wie es aussieht. Dann kann ich dir auch gleich die Effekte zeigen, die da auftreten.“

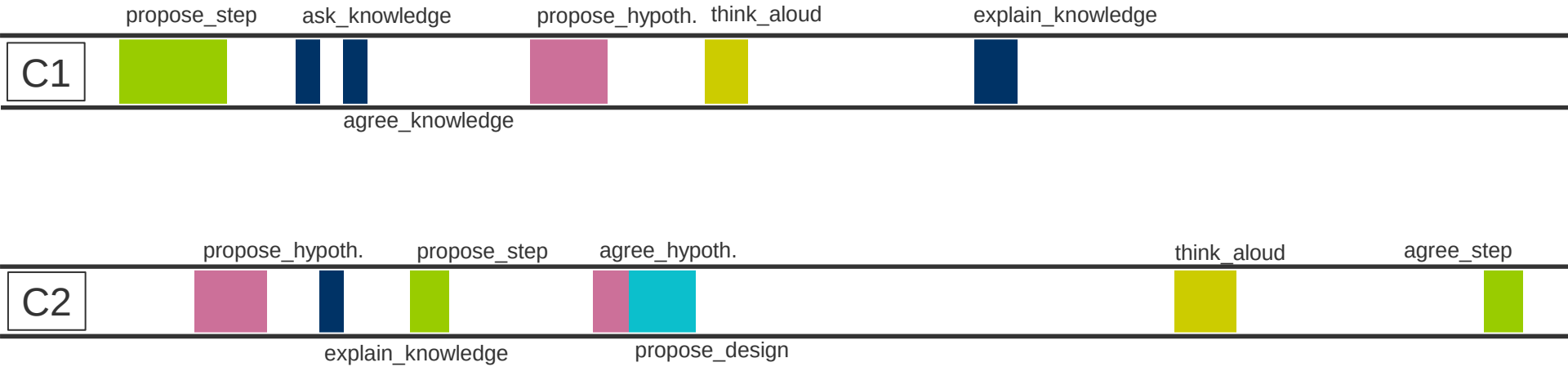
Beispiel: CA1 0:15:20 - 0:17:12



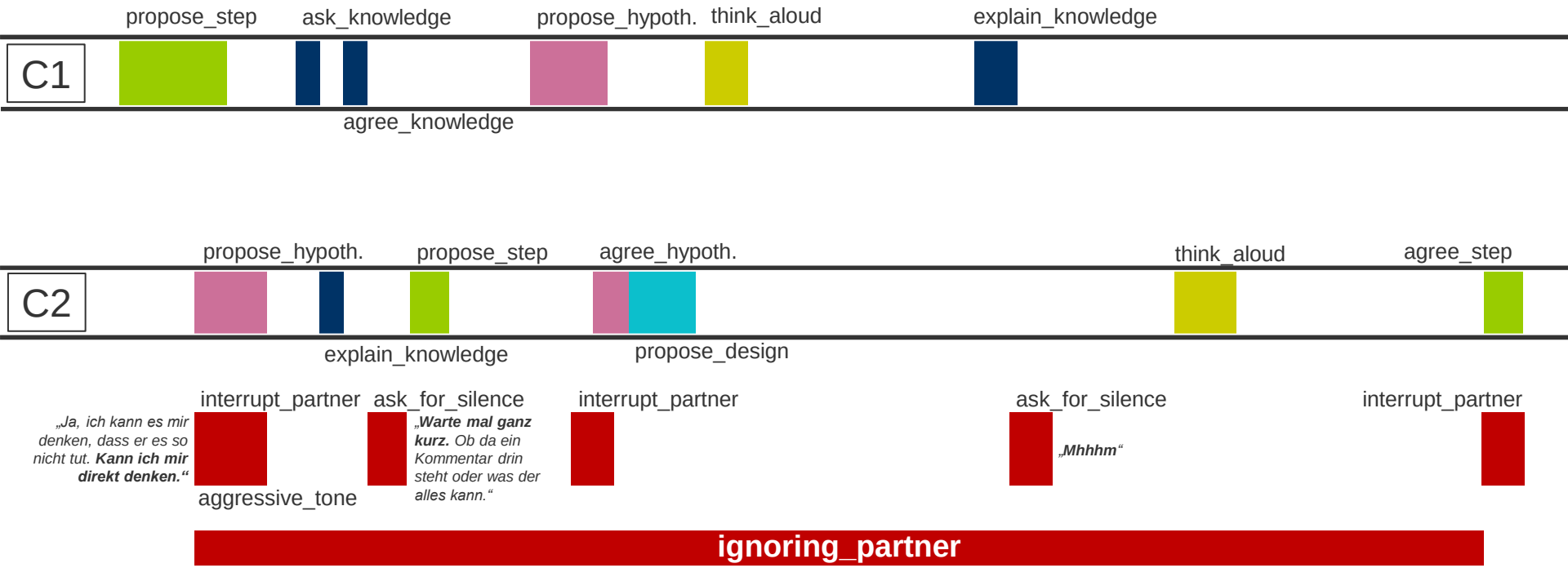
„Du kannst ja mal einkommentieren und dann schauen wir uns in der Demo an, wie es aussieht. Dann kann ich dir auch gleich die Effekte zeigen, die da auftreten.“

„Ja, ich kann es mir denken, dass er es so nicht tut. Kann ich mir direkt denken.“

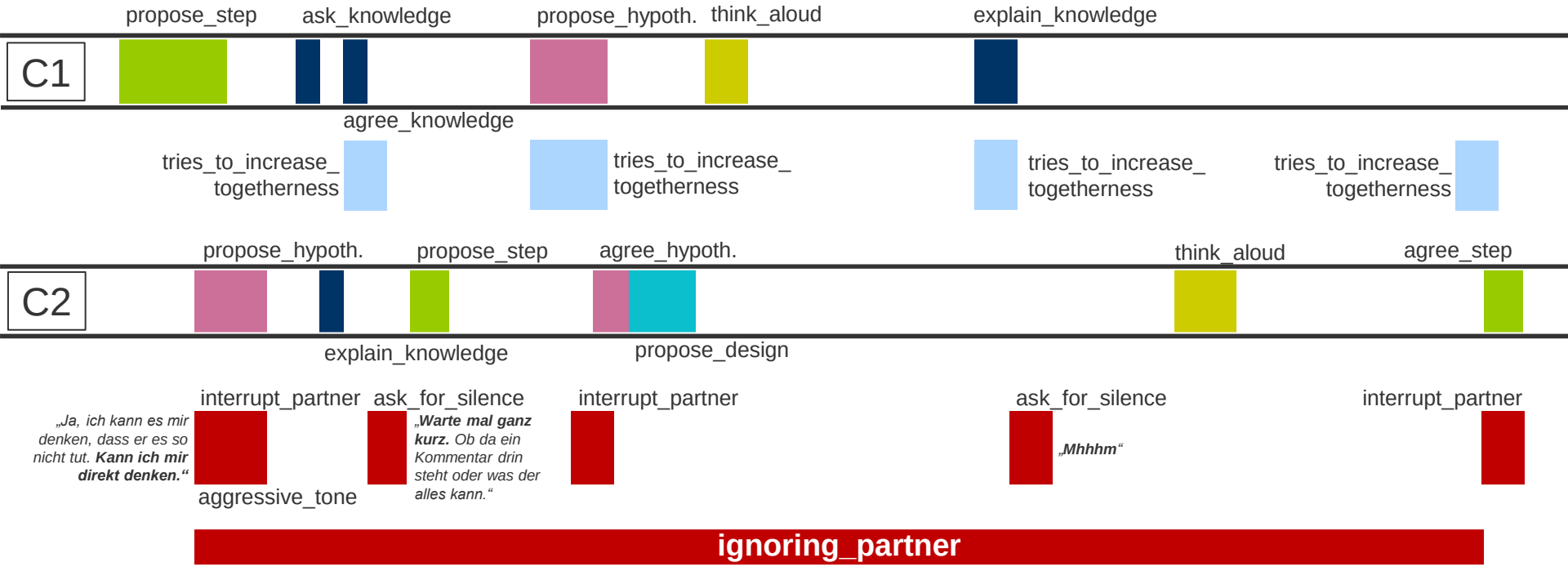
Beispiel: CA1 0:15:20 - 0:17:12



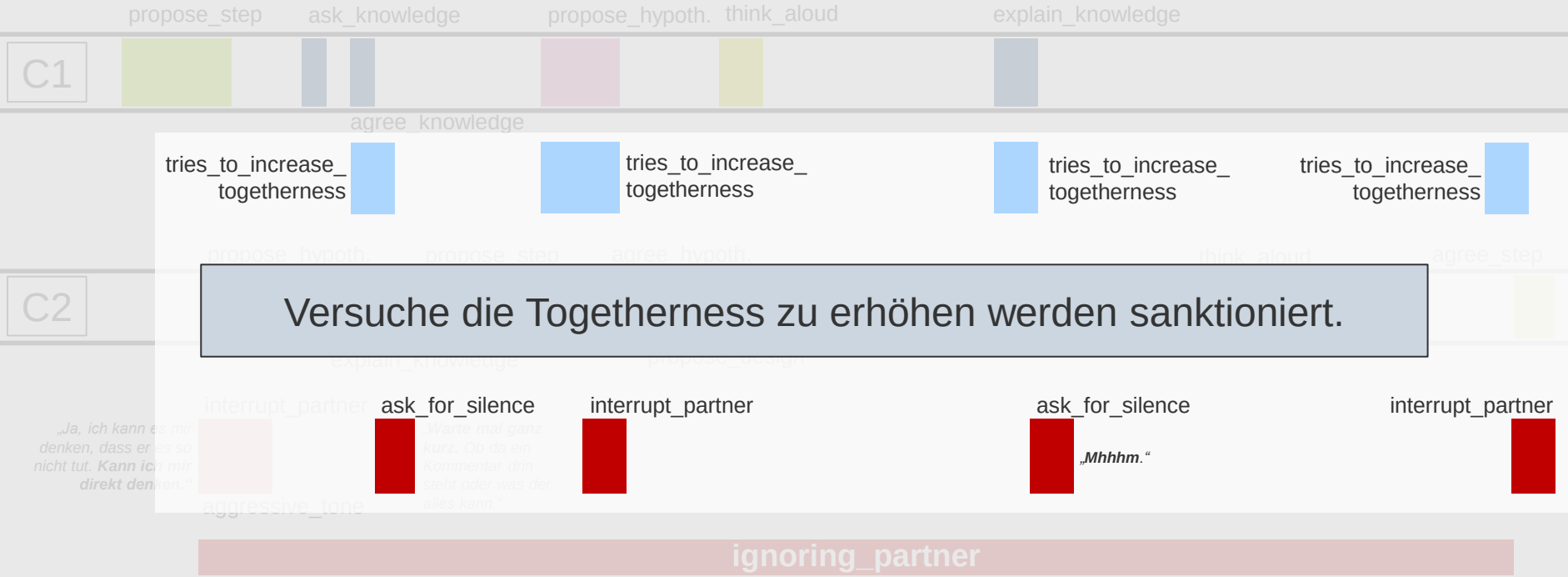
Beispiel: CA1 0:15:20 - 0:17:12



Beispiel: CA1 0:15:20 - 0:17:12



Beispiel: CA1 0:15:20 - 0:17:12



Beispiel CA1: Weitere Auffälligkeiten

C1 ist jeweils nur zum Commiten als Driver tätig
C2 reißt die Tastatur, sobald es komplexer wird, an sich.

Lediglich auf eine Entscheidung zum GUI-Design hat C1 einen Einfluss. Ansonsten keinen Faktor bei Entscheidungen.

Wissen wird aufgrund des Verhaltens von C2 und dem in die Passivität gedrängten C1 kaum vermittelt.

Vorteile der Paar-Programmierung gehen durch das autoritäre Verhalten von C2 verloren.

Beispiele aus Sitzung PA3

	Driver: P3 Driver für die gesamte Sitzung		Observer: P1 Meist zurückgelehnt dasitzend

Entwicklung eines API-Endpoints zum Anzeigen von Daten im Frontend.
Die Daten sind als Prozentwerte gegeben und sollen im Frontend als Werte zwischen 0 und 1 angezeigt werden.

Zum aktuellen Zeitpunkt ist die Funktionalität bereits fertig und sie haben gerade die Einrückung zur besseren Lesbarkeit angepasst.
P1 macht einen Verbesserungsvorschlag…

Beispiel PA3: Weitere Auffälligkeiten

P1 neigt zu einem belehrenden Tonfall.

P1 sitzt die meiste Zeit zurückgelehnt und gibt P3 sehr genaue Anweisungen, was zu tun ist.

P3 ist an einigen Stellen (hier am deutlichsten) genervt von dem Verhalten von P3 und beschwert sich nach der Sitzung darüber.

Ungefragte Belehrung von P1, führt zu Anspannung / Genervtheit von P3 und damit u.a. zum Treffen einer schlechten Entscheidung.

Beispiele aus Sitzung AA1

Observer: A1
Observer für gesamte Sitzung.

Driver: A2
Driver für gesamte Sitzung.

Beseitigen von Inkonsistenzen in 5 unterschiedlichen Listen im Frontend. Arbeiten an dem Anzeigen des richtigen Icons. Dafür wird der Name der Grafik u.a. in Abhängigkeit ob das Objekt aktiv/inaktiv ist zusammengebaut.

Haben die Methode an der sie gerade arbeiten komplett neu geschrieben. Und sich u.a. währenddessen darauf geeinigt, dass sie dafür ein CMMicroObject brauchen. A2 ändert das nun zu einem CMMiniObject.

Beispiele AA1: Weitere Auffälligkeiten

A1 neigt ähnlich wie P1 zu einem belehrenden Tonfall gepaart mit sehr aggressiven Interventionen.

A1 ist ähnlich wie P1 die ganze Zeit Observer und gibt spezifische Anweisungen.

Anders als P3, lässt sich A2 nicht aus der Ruhe bringen (bis auf 1x innerhalb der Sitzung als er A1 bittet ihn ausreden zu lassen).

Warum hat das Verhalten von A1 hier keinen schädlichen Einfluss?

Beispiele AA1: Weitere Auffälligkeiten

A1 neigt ähnlich wie P1 zu einem belehrenden Tonfall gepaart mit sehr aggressiven Interventionen.

A1 ist ähnlich wie P1 die ganze Zeit Observer und gibt spezifische Anweisungen.

Anders als P3, lässt sich A2 nicht aus der Ruhe bringen (bis auf 1x innerhalb der Sitzung als er A1 bittet ihn ausreden zu lassen).

Warum hat das Verhalten von A1 hier keinen schädlichen Einfluss?

Forschen mit der Grounded Theory Methodology:

Wie entstehen Konzepte & Hypothesen?

Forschen mit der Grounded Theory Methodology:

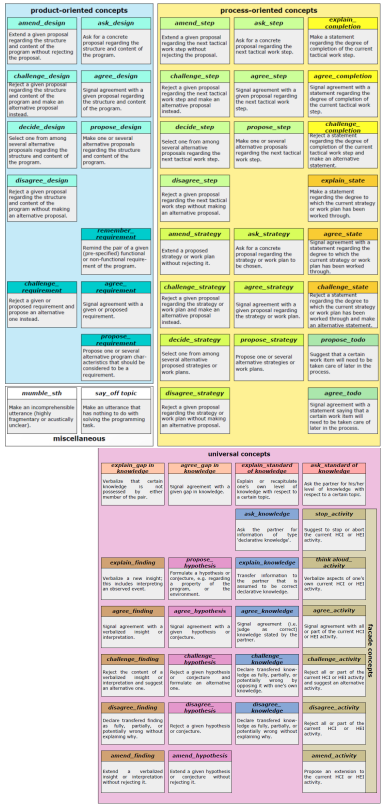
Nuancen der Base-Layer Konzepte

Entscheidungsepisoden beginnen mit **propose_*** , gefolgt von **disagree_*** ; **challenge_*** ; bis zu einem **agree_***

Es gibt unterschiedliche Arten des propose, challenge, disagree, agree:

- Mit oder ohne Begründung
- Spezifisch oder Abstrakt
- Vorschlag durch „Vormachen“
- „Zustimmung“ durch keinen Widerspruch

Die Rollenverteilung (Driver/Observer) hat großen Einfluss auf den Entscheidungsprozess.



Driver-Observer-Dynamik im Fokus

Der Driver hat mehr Möglichkeiten Entscheidungen zu initiieren und hat automatisch das letzte Wort.

Der Observer kann intervenieren. Und ist gefährdet den Anschluss zu verlieren, wenn er dies nicht tut.

Gleichberechtigung ist für gute Entscheidungsprozesse fundamental (Behauptung!). Driver muss dies aktiv ermöglichen.

Betrachtung von CA1 unter diesen Gesichtspunkten, führte zum Fokus auf autoritäre Partner, die die Togetherness aktiv gefährden.

Autoritäres Verhalten in der PP

Verhalten von autoritären Partnern:

-  Unterbrechen (und mit anderem Thema fortfahren) — CA1/PA3/AA1
-  Ignorieren — CA1
-  Belehren — CA1/PA3/AA1
-  Aggressiver Tonfall — CA1/AA1

Auswirkungen auf die Sitzungen:

-  Observer wird in die Passivität gedrängt / geringe Togetherness — CA1
-  Driver reagiert genervt + schlechte Entscheidung — PA3

Weitere Ideen und Überlegungen

Fallstudie zu JA1 – JA9:
- 9 aufeinander aufbauende Sitzungen ohne Unterbrechung.
- Von Design bis zum finalen Produkt
- Entscheidungen können bewertet werden

Entscheidungsausgänge:
- Ursprünglicher Vorschlag
- Gegenvorschlag
- Gemeinsamer Kompromiss
- Entscheidung vertagt

Entscheidungstreiber
- Wissenstransfer
- Argumentation
- Third-Party

Einfluss des Senior-Junior-Verhältnis auf die Entscheidungs-findung:
- Wann setzt sich wer durch?

Fazit

Schädliche Auswirkung von autoritären Verhalten auf die Paar-Programmierung.

- Erste Erkenntnisse und Belege
- Aber: Nicht eindeutige Auswirkungen → Es fehlen Daten!

Einfluss der Driver-Observer-Dynamik auf die Entscheidungs-findung.

- Erste Erkenntnisse und Belege
- Aber: Wie geht es damit weiter?

Fazit

Schädliche Auswirkung von autoritären Verhalten auf die Paar-Programmierung.

- Erste Erkenntnisse und Belege
- Aber: Nicht eindeutige Auswirkungen → Es fehlen Daten!

Einfluss der Driver-Observer-Dynamik auf die Entscheidungs-findung.

- Erste Erkenntnisse und Belege
- Aber: Wie geht es damit weiter?

Vielen Dank!

Fragen / Anregungen / Kommentare?

Quellen

[1] **Dijkstra, Edsger W.** "The humble programmer." *Communications of the ACM* 15.10 (1972): 859-866.

[2] **Zieris, Franz.** *Qualitative analysis of knowledge transfer in pair programming*. Diss. 2020.

[3] Stack Overflow Developer Survey 2018

[4] Stack Overflow Developer Survey 2020

[5] **Arisholm, Erik, et al.** "Evaluating pair programming with respect to system complexity and programmer expertise." *IEEE Transactions on Software Engineering* 33.2 (2007): 65-86.

[6] **Zieris, Franz, and Lutz Prechelt.** "PP-ind: A repository of industrial pair programming session recordings." *Context* 50 (2020): 9.

[7] **Salinger, Stephan, and Lutz Prechelt.** *Understanding Pair Programming: The Base Layer*. BoD–Books on Demand, 2013.

Bildquellen

1.1570789.figures.f1.jpeg (650×826) (scitation.org)