31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Das

Paarungsverhalten

qualitätsbewusster

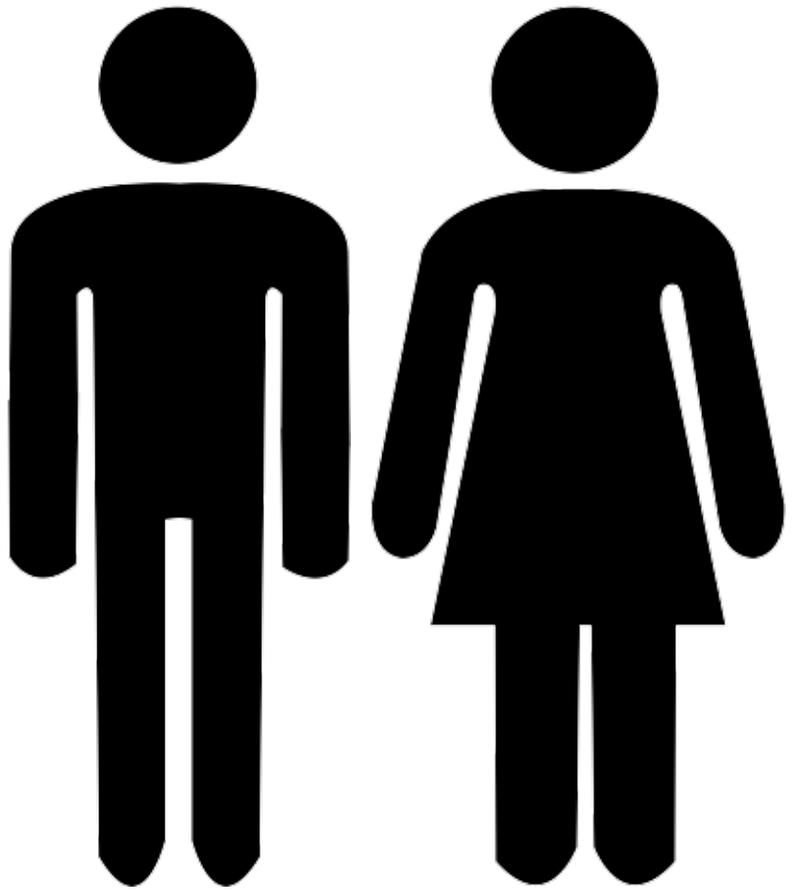
Softwareentwickler

Pair Programming Coaching im Großprojekt

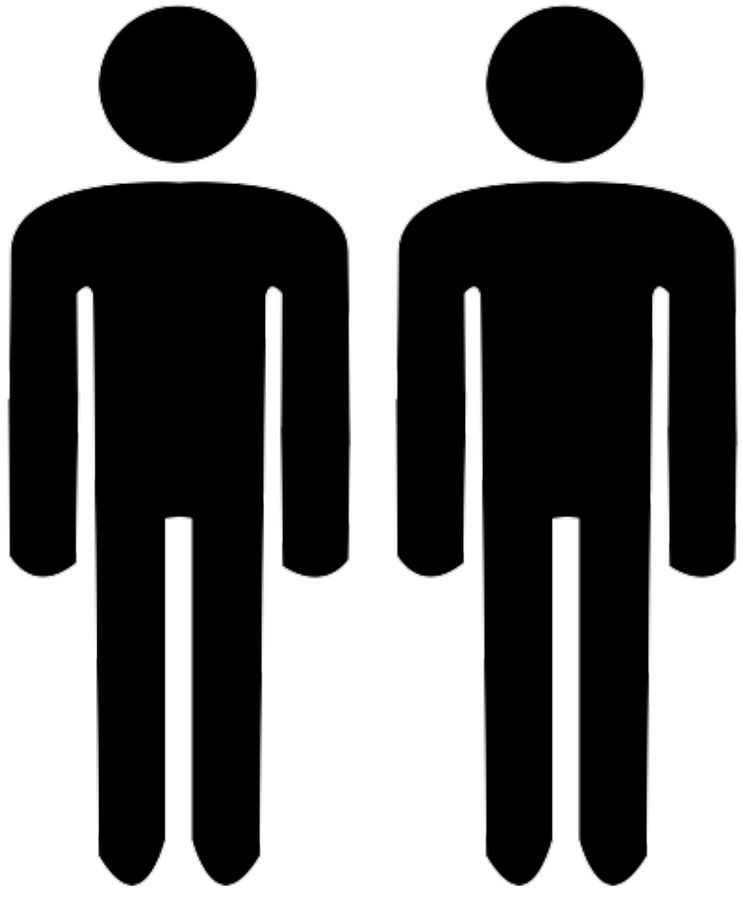
Thomas Much

muchsoft.com

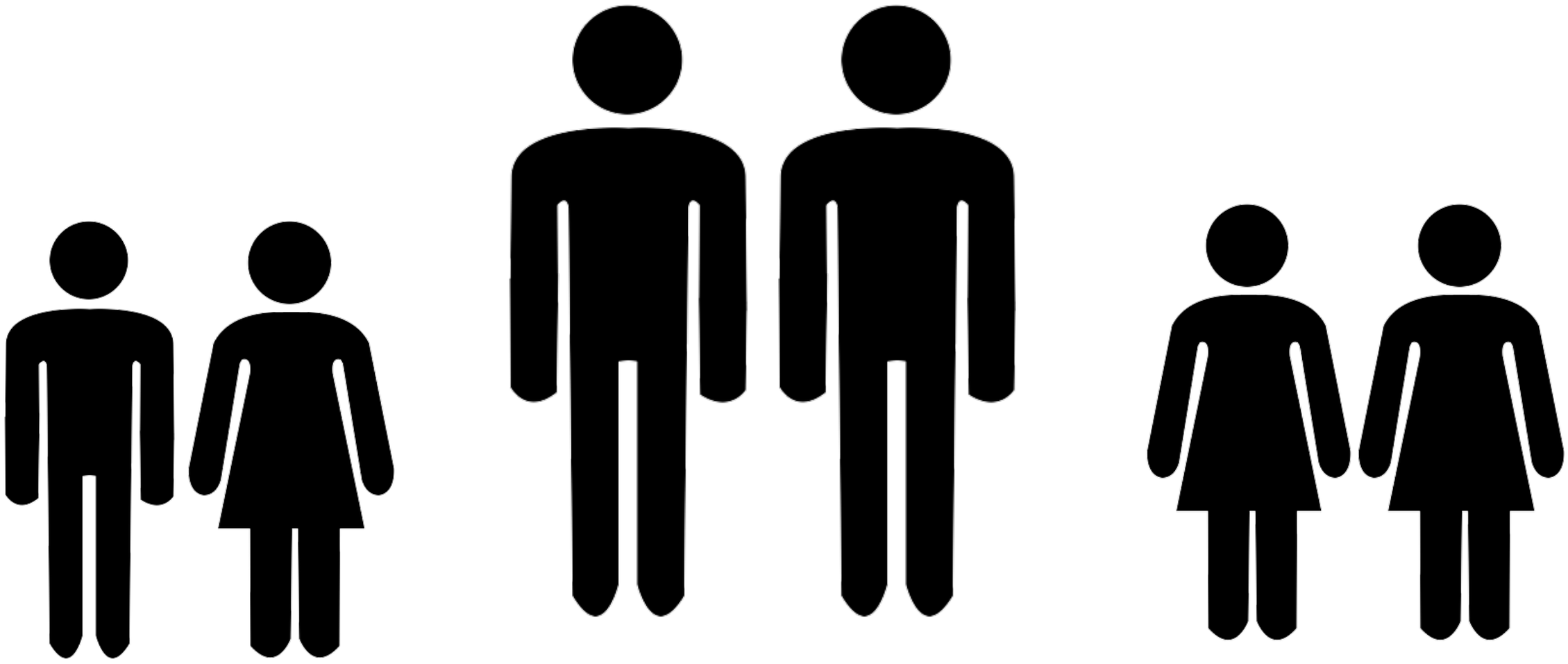
Das Paarungsverhalten …



… in Softwareprojekten …



… wird bunter!



Es geht um

- Teams
- Zusammenarbeit
- Kommunikation

speziell im Bereich der *Softwareentwicklung*.

Agile Softwareentwicklung

- Scrum und Kanban sind etabliert.

- Hilfsmittel:
  - Continuous Integration bzw. Delivery bzw. Deployment
  - Test-Driven Development (TDD)
  - Pair Programming

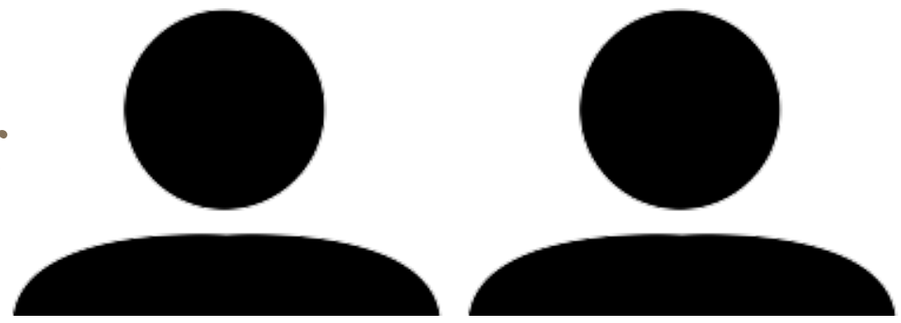
- Pair Programming ist eher ein Geheimtipp
  - ignoriert, abgelehnt – oder heiß geliebt

- Dieser Vortrag ist ein *Praxisbericht* über erfolgreiches Pair Programming.

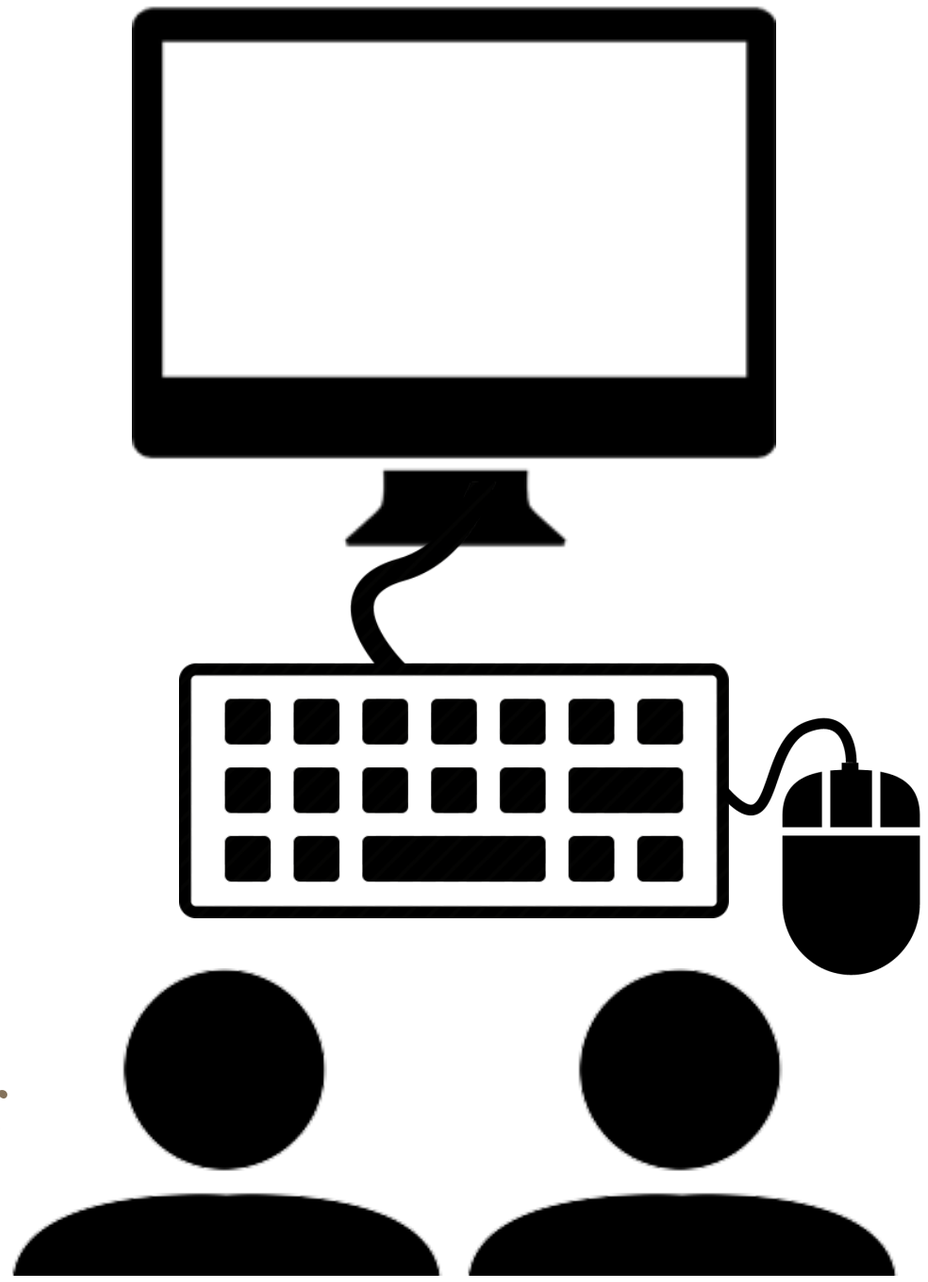
Was ist Pair Programming?

Was ist Pair Programming?

Zwei Programmierer



Was ist Pair Programming?

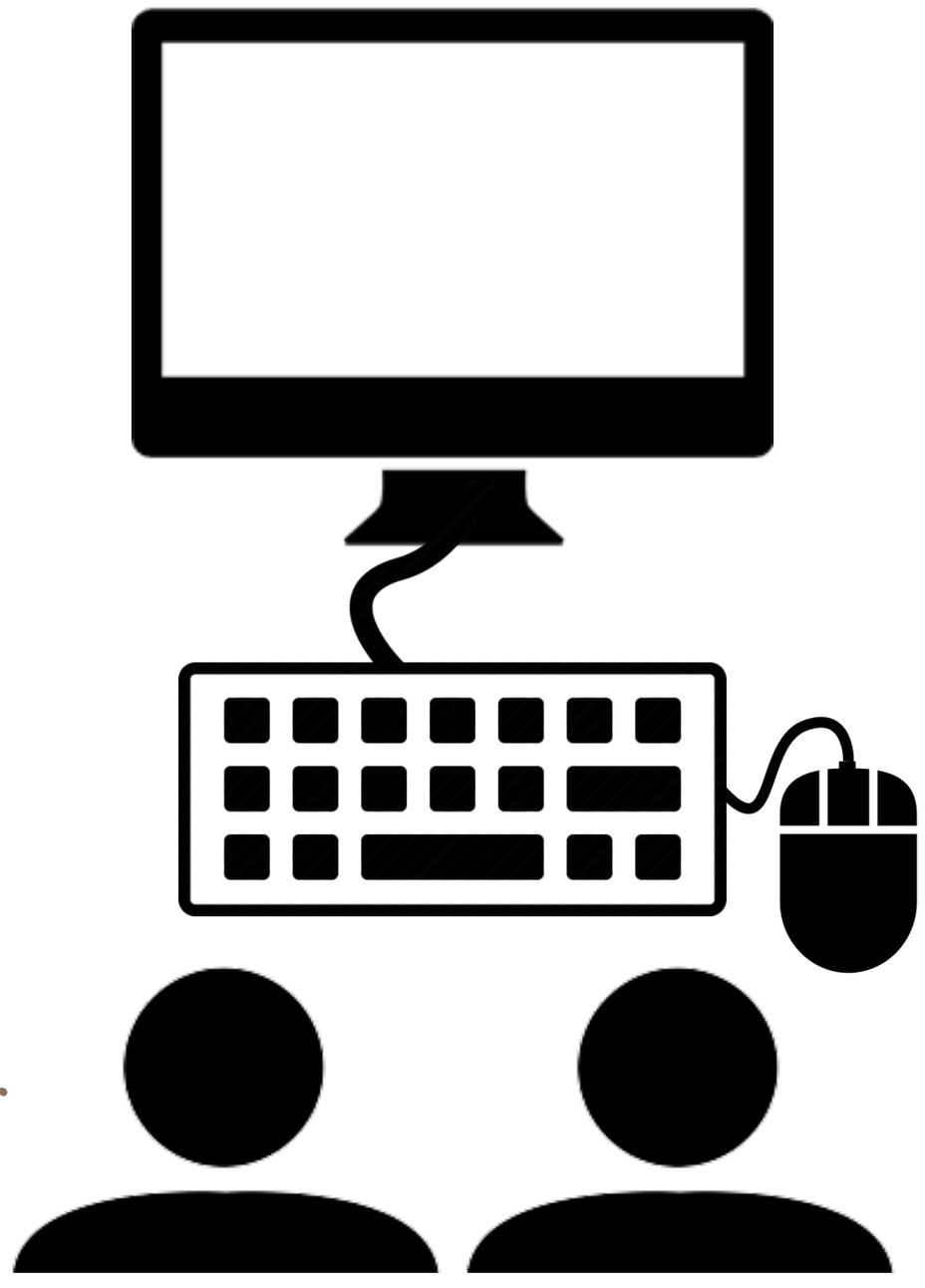


Zwei Programmierer

arbeiten zusammen
am selben Rechner

Was ist Pair Programming?

Zwei Programmierer

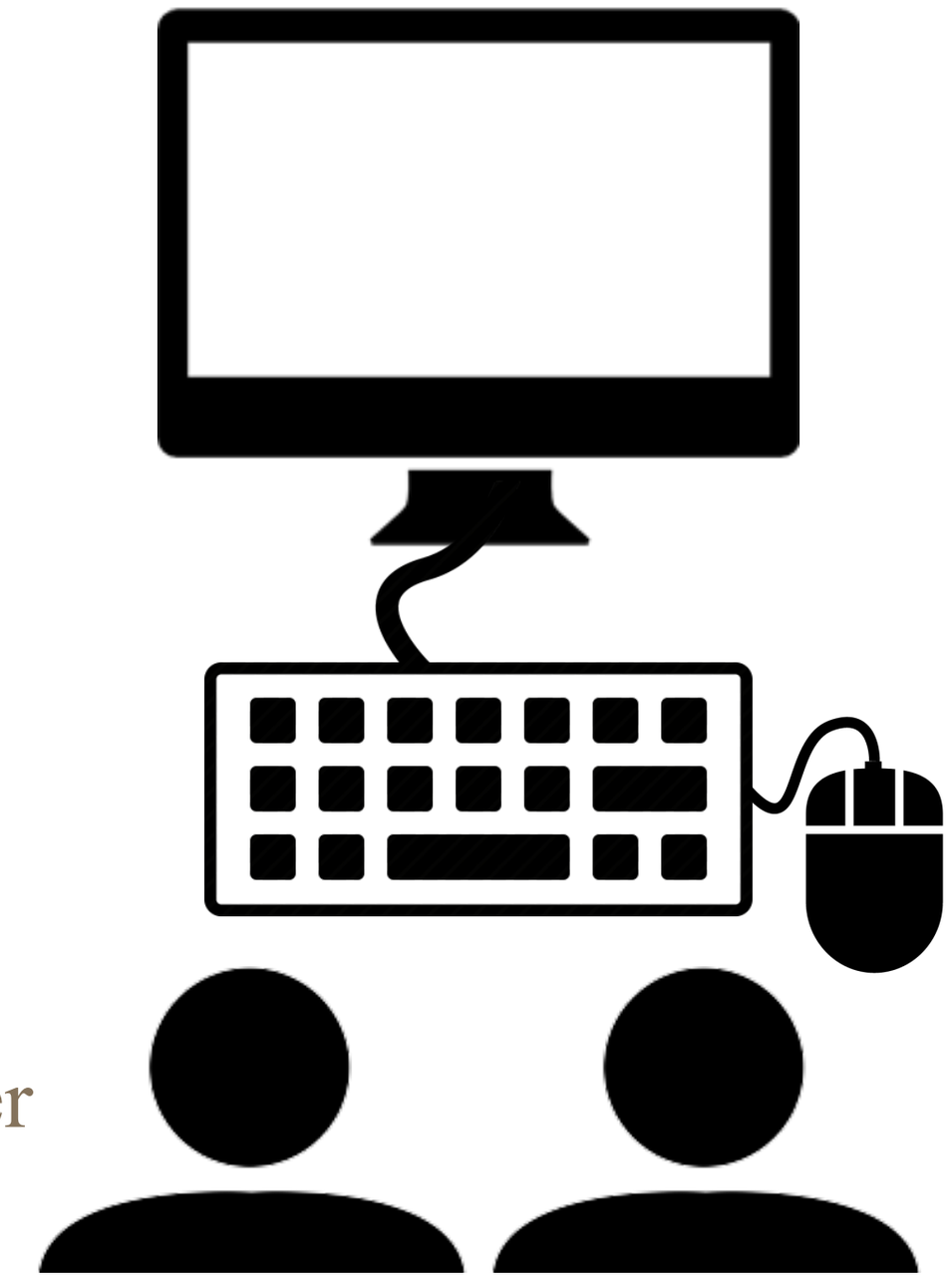


arbeiten zusammen
am selben Rechner

an einem geeigneten
Arbeitsplatz

Was ist Pair Programming?

Zwei Programmierer



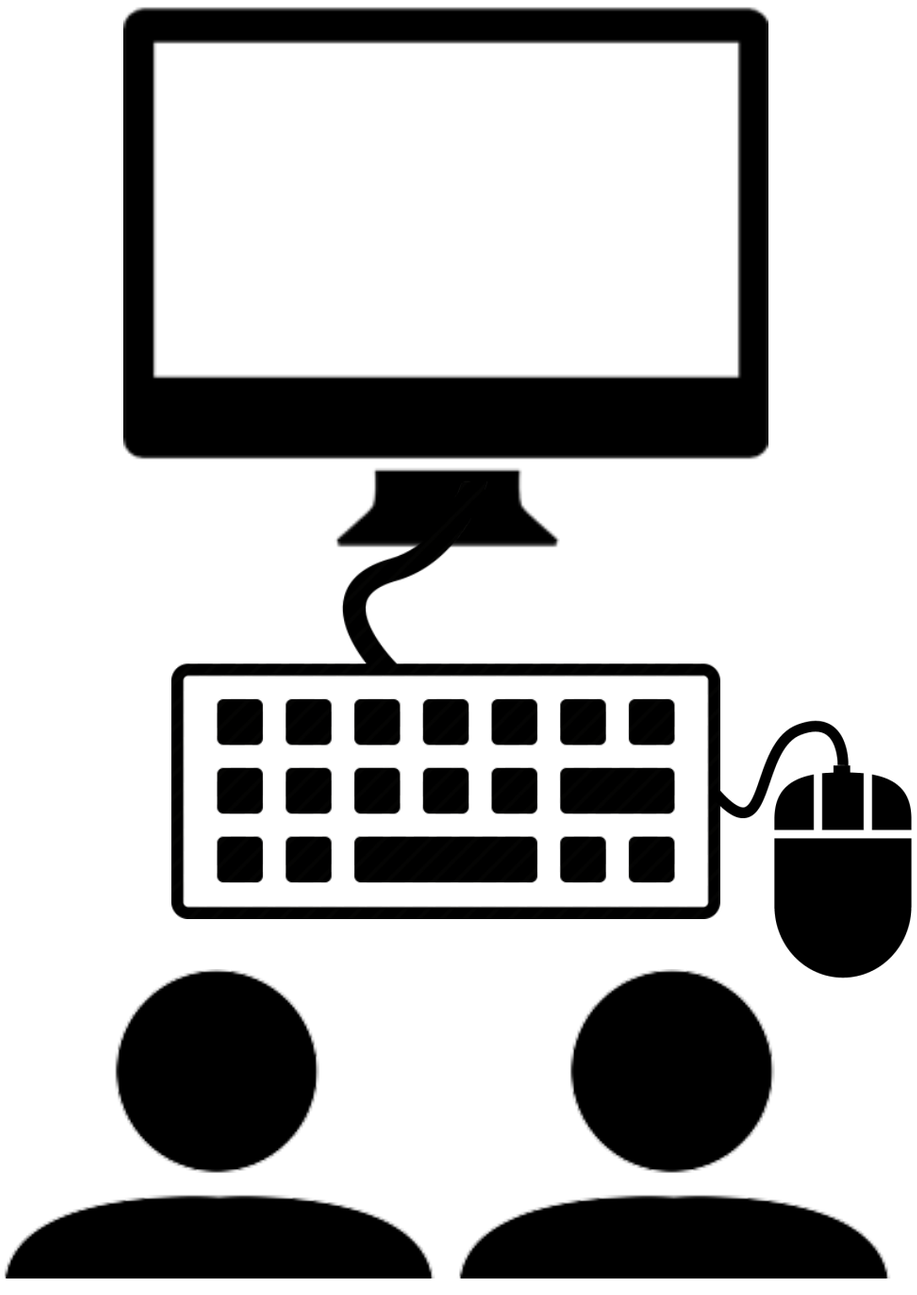
arbeiten zusammen
am selben Rechner

an einem geeigneten
Arbeitsplatz

in zwei Rollen:

- **Driver**
- **Navigator**

Was ist Pair Programming?

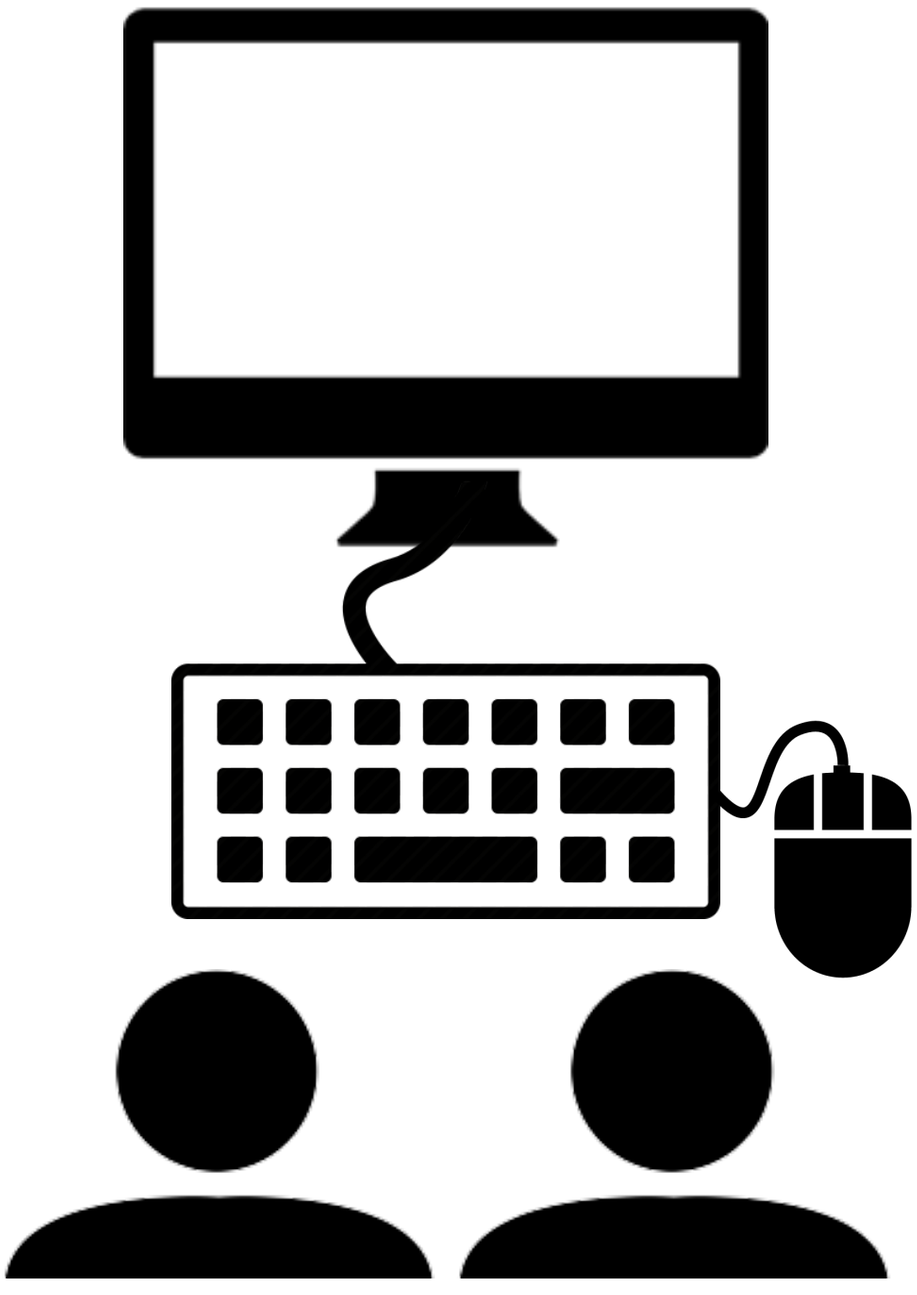


- Zwei Rollen:
  Driver & Navigator

+Häufige Rollenwechsel!

+Effiziente Pausen!

Was ist Pair Programming?



- Zwei Rollen:
  Driver & Navigator

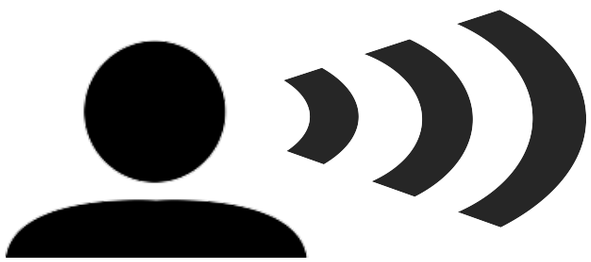
+Häufige Rollenwechsel!

+Effiziente Pausen!

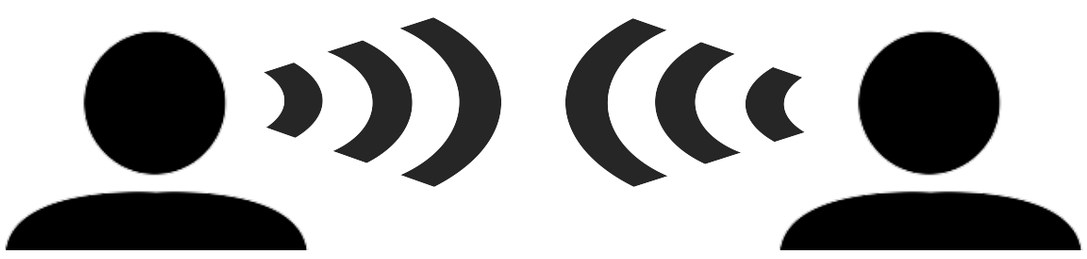
+Regelmäßige Pair-Wechsel

+Pairing-fähige Aufgaben
(gut geschnittene Stories & Tasks)

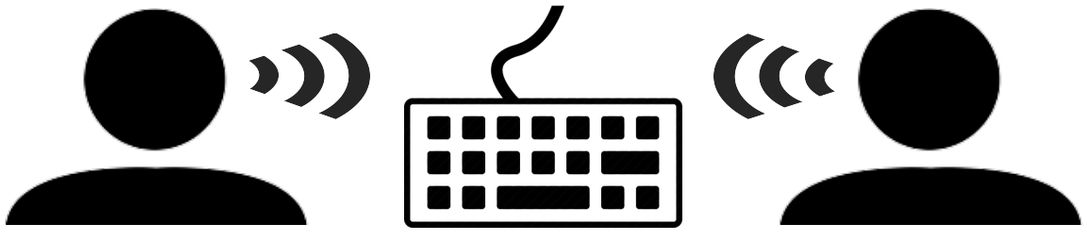
Was ist Pair Programming, was nicht?



Reden & "berieseln lassen"



Brainstorming
Know-How-Transfer
Review



Pair Programming

Pair Programming

Stop Talking, Start Coding

Aber trotzdem die Kommunikation nicht vergessen!

Also:

Richtiges Pair Programming muss gelernt werden!

Einfach nur zwei Programmierer
vor einen Rechner zu setzen,
kann absolut kontraproduktiv sein.

Aber wie lernt man Pair Programming?

Ein paar Ideen aus dem Projektalltag.

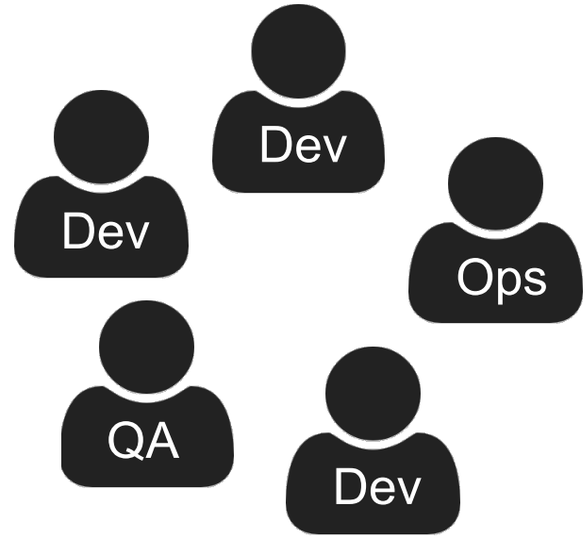
Projekt – Rahmenbedingungen

- Eines der größten E-Commerce-Projekte in den vergangenen Jahren in Deutschland
- > 10 Teams
- > 200 Projektmitarbeiter
- Vertikale (fachliche) Schnitte
- Shared-Nothing-Architektur

Projekt – Rahmenbedingungen

- Eines der größten E-Commerce-Projekte in den vergangenen Jahren in Deutschland
- > 10 Teams
- > 200 Projektmitarbeiter
- Vertikale (fachliche) Schnitte
- Shared-Nothing-Architektur
- Siehe Entwickler-Blog *dev.otto.de*

Projekt – Teams und Technologien



- Viele Teams
- Jedes Team besteht aus Devs, QA, Ops

- Jedes Team hat technologische Entscheidungsfreiheit
- Java-VM ist aber bei den meisten die Grundlage

- Viele Technologien
- Java, Scala, Clojure, JavaScript, Ruby, Swift …

Das Projekt schreitet voran

- Teams sind agil, qualitätsbewusst, motiviert
- Prozess liefert häufige (Live-)Deployments

- Aber nach mehreren Monaten
  - Gefahr durch Routine
  - Gefahr von Know-How-Verlust durch wechselnde Team-Mitglieder

- Wie halten wir das Wissen und die Qualität aufrecht?

Idee

Pair Programming durch *Coaching* aktiv fördern

(aber bitte nicht erzwingen!)

Warum Pair Programming fördern?

- Pair Programming fördert implizit die Werte für *effiziente* Team-Arbeit:

- **Vertrauen**
- **Kommunikation**
- **Selbstreflexion**
- **keine Überheblichkeit**

Ziele aus Unternehmenssicht

- Know-How-Transfer
  - im Team und idealerweise auch zwischen den Teams
  - Best Practices!

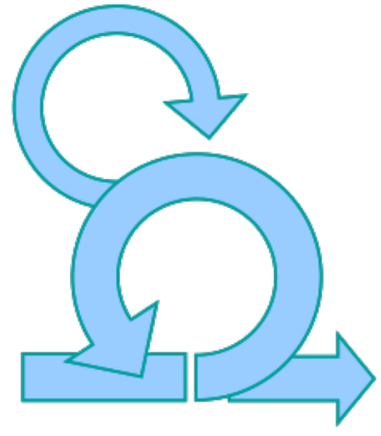
- Weitere Qualitätssteigerung

- Nachhaltige Softwareentwicklung

- Spaß am effizienten Arbeiten

Das alles liefert *richtig gelebtes* Pair Programming.

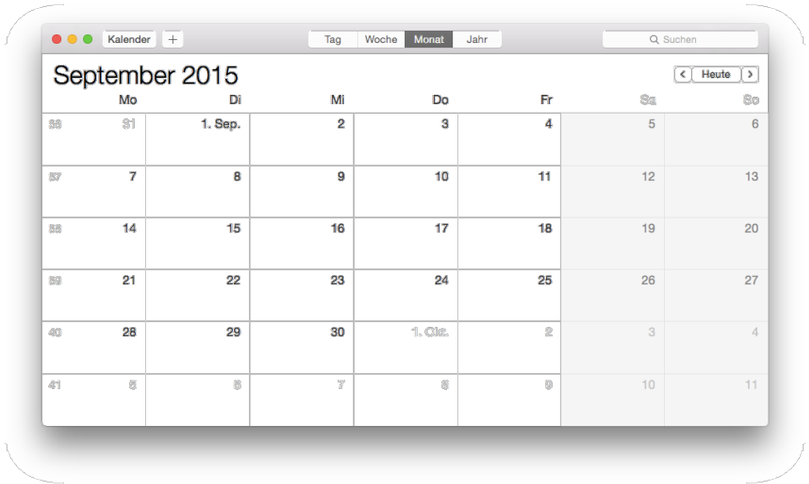
Ziele für das Coaching



- Keine langen Workshops.
- Learning-by-Doing.
- Bestehendes Pairing (und mögliche Probleme) bewerten, Verbesserungen aufzeigen und *üben*
- Blick von außen, Spiegel für das Team
- *Sprint-Ziele müssen erreicht werden!*

Was ist Pair Programming Coaching?

- Team-Begleitung für ca. zwei Sprints (insgesamt ca. vier Wochen)



- Bewusst *externer* Coach
  - Kein interner Mitarbeiter z.B. aus anderem Bereich
- Für die Zeit des Coachings *vollwertiges Team-Mitglied* als Softwareentwickler!
- Externer Blick auf Team-Abläufe, Technologien und Architektur

Pair Programming Coaching vs. Agile Coaching

- Fließender Übergang zum agilen Coaching, aber *Ansatz-/Ausgangspunkt ist ein entscheidend anderer!*
- Nicht der ~~Prozess~~Ablauf steht im Mittelpunkt, sondern der *Code*
- "Wie realisieren wir ein Feature so, dass der *Code nachhaltig* ist?"
- Nicht das Entwicklungsvorgehen ist Garant für guten Code, sondern *guter Code ermöglicht einen ordentlichen Ablauf.*

Pair Programming nicht erzwingen

- Kein dogmatisches Pair Programming.
- Teams dürfen auch anders arbeiten, um *nachhaltige* Software zu entwickeln.
- Nachhaltige Software:
  Wartbar, erweiterbar, ersetzbar, fehlertolerant, dokumentiert, Wissen darüber in den Köpfen verteilt…
- Pair Programming ist nur ein Vorschlag.
- Löst aber obige Anforderungen recht gut 😉

Warum Coaching und kein Workshop?

- Typische Umfrage einige Zeit nach einem Pair-Programming-*Workshop:*
  - "Och nö"
  - "Keine Lust"
  - "Ist irgendwie wieder eingeschlafen"
  - "Haben keinen besseren Code mit weniger Fehlern entwickelt"
  - "Alleine bin ich schneller"
- Daher längere *Coaching-Begleitung!*
- Teams wollen oft Pair Programming anwenden, brauchen aber immer mal wieder Hilfe/Motivation

Coaching statt Workshop

- Pair Programming ist Gewöhnungssache
  - Für das Management sowieso
    ("2 statt 1 Entwickler für eine Aufgabe? Was das kostet!" –
    "Oh, die Velocity ist ja gar nicht gesunken")
  - Aber auch ungewohnt für viele Entwickler
    ("Ich will lieber alleine frickeln" etc.)
- Wer es erfolgreich *erlebt* hat, will es nicht mehr missen.

Coaching – Ablauf

1 Sprint
(2 Wochen)

- Kickoff-Meeting
- Kennenlernen/Einarbeiten und Pair Programming mit dem Coach üben
- Zwischenstand

ca. 1 Sprint
(1-2 Wochen)

- evtl. eine / ein paar Wochen Pause
- Coach achtet auf aufgefallene Probleme
- Üben, üben, üben
- Coaching-Retro

ca. 1 Woche

- evtl. nach ein paar Wochen/Monaten:
- Berichten, beobachten, helfen, üben

Wann sollte das Coaching stattfinden?

- Nötig/gewünscht meistens so schnell wie möglich
- Wirklich passen tut es den Teams aber fast nie…

- *Hier:*
- Beginn mitten im laufenden Großprojekt
- In der heißesten Phase –
  erstes Release (MVP) wenige Monate später

Erforderliche Vorbereitungen

- *Team:*
- Pairing-fähige Tickets/Tasks vorbereiten/raussuchen
- *Coach:*
- Vorher Technologien erfragen und grob einarbeiten (z.B. Programmiersprache)

Zielvorgaben

- Ziel sollte *nicht* sein, während des Coachings zwingend *alles* im Pair zu entwickeln.
- Lieber *möglichst geeignete Tasks* im Pair.
- Coach sollte aber in dieser Zeit immer einen Pairing-Partner haben.
- *Pair Programming wird dadurch "nebenbei" etabliert.*

Wichtige Randbedingungen

- Coach muss *Team-Mitglied* sein

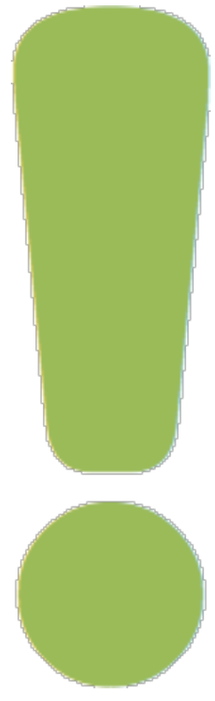
- Coach braucht *Vertrauen!*
  - Vom Team und vom Management
  - Keine inhaltliche und personelle Berichtspflicht an das Management
    (Management muss im Zweifel mit dem Team selber reden)

- Coach muss *programmieren* können

- Coach sollte technischer Sparrings-Partner sein
  - So macht Pair Programming am meisten Spaß
  - Wichtig für den Start!

Coach als Entwickler

- Agiles Coaching häufig durch eher methodische Coaches
  - Mit Schwerpunkt auf Team / Prozess / Ablauf
  - Absolut sinnvoll, um agiles Vorgehen zu etablieren!
- Programmierern fehlt aber oft der wichtigste Aspekt: **Programmierer wollen programmieren.**
  - Prozess ist für sie notwendiges Übel ("überflüssige" Bürokratie)
- *Ein "Techie"-Coach, der mitcoden kann, wird ganz anders aufgenommen.*



Herausforderungen (1)

- Starke Charaktere, die Lösungen schnell selbst hinbekommen
  - Müssen verstehen lernen, dass eine Lösung dem Team gehören sollte, sonst ist sie i.d.R. nicht nachhaltig

- Eher introvertierte Charaktere
  - Müssen doch mal mitmachen, um zu erfahren, dass man auch mit gelegentlichem Pairing viel erreichen kann

- "Pair Programming lohnt sich hier nicht, weil …
  - … es hier nicht viel zu programmieren gibt."
  - … hier kaum Tests zu schreiben sind."

Herausforderungen (2)

- Team macht Pair Programming
  - noch gar nicht
  - komplett, aber mit Pairing-(Wissens-)Inseln

- Ineffiziente / keine / zu wenig Pausen und Wechsel

- Entwicklungsprozess liefert keine geeigneten Tickets
  - Stories/Tasks richtig geschnitten?

- Digitales Board
  - Fehlende Dynamik; Pair-Avatare nicht offensichtlich

Herausforderungen (3)

- Team ist noch keine Einheit
- Team ist nach Teamveränderung keine Einheit mehr

https://de.wikipedia.org/wiki/Teambildung#Phasenmodell_nach_Tuckman

Mehr als Pairing – The Team ~~Healer~~Helper

- Ab und zu kommen auch die besten Teams ins Stocken

- Coach kann hier helfen
- Aus dem Blickwinkel Pair Programming
  - Was hindert uns am Pair Programming?

- *Hilft oft, den agilen Ablauf wieder in Ordnung zu bringen!*
  - Blick von Außen hilft dem Team, sich *selber* zu helfen

- Coach muss aber aufpassen, funktionierende Teams nicht kaputt zu machen!

Weitere Angenehme Nebenwirkungen

- Coach kann auf Wunsch einen Blick auf die ganze Technologie/Architektur des Teams haben
  - Liefert oft wertvolle Ideen und Tipps als anfangs Außenstehender
- Schutz gegen Störungen
  - "Du, kannst Du mal schnell eben…"
  - Leider kein Schutz gegen zu viele Meetings… 😐
- Coaching fördert *Selbstheilung* des Teams
  - solange die Probleme klein sind
  - bevor etwas eskaliert und die Aufmerksamkeit des Managements benötigt

XT*-Pairing: Entwickleraustausch

- Cross-*Team*
  (Abteilungs-/Konzern-Best-Practices austauschen)

- Cross-*Technologie*
  (riesiges Potenzial für neue/kreative Ideen)

- Wichtig, wenn z.B. durch Microservices die Teams kleiner werden!

- Problematisch bzgl. Ressourcenplanung
  - Daher idealerweise ganze Sprints, gerne auch mehrere
  - Kann dann wie Urlaub geplant werden

Interdisziplinär

- DevOps! *(gilt das noch als interdisziplinär?)*
- DevQA! *(gilt das noch als interdisziplinär?)*
- DevDesign (DevUX)?

- Ops+QA eigentlich sowieso Teil agiler Teams
- Echtes Pair Programming ist hier aber nur sinnvoll, wenn die Ops'ler bzw. QA'ler programmieren können
- Sonst ist das eher Brainstorming, Review etc.

Warum macht dann nicht jeder Pairing?

- Vorbehalte im Management
  - Kurzfristig kann das Zeit (und damit Geld) kosten – auch wenn sich das mittel- und langfristig rechnet
- Vorbehalte in den Teams
  - Coole (extrem gute) Entwickler, die einfach nur ihr Ding machen wollen
  - Nicht so coole Entwickler, die sich plötzlich nicht mehr im Team verstecken können
- Erfordert *Mut* bei den Entscheidern, sich für Pair Programming zu entscheiden (bzw. das zu ermöglichen)
- Erfordert *Willen* zur Teamarbeit bei den Entwicklern

Irgendwie "teamfähig" war gestern

- Wir brauchen spezielle Skills für die SW-Entwicklung!

Position:
Agiler Java-Entwickler (m/w) (Full-Stack) für Hamburg gesucht

Aufgabe:
Unterstützung unseres agilen Scrum-Softwareentwicklungsteams bei der Weiterentwicklung der Fachdomäne "Bestellen".

Qualifikation:
. Sehr gute Kenntnisse in der Webentwicklung mit Java (Full-Stack: Java SE, JavaScript, CSS, HTML) und gängigen Frameworks wie Sprin
. Gute Kenntnisse in Scala, REST, Testframeworks (bspw. Mockito, Selenium) und idealerweise auch NoSQL-Datenbanken (bspw. MongoD
. Wünschenswert sind Erfahrungen mit Responsive Design Webanwendungen, Micro Services, Akka, Jenkins, sbt, Gradle und SoapUI
. Teamfähigkeit und gute kommunikative Fähigkeiten sind ein absolutes Muss.
. Die Beherrschung der gängigen Entwicklungstools wie IDE, Jenkins, Git usw. setzen wir voraus.
. Erfahrung in und Affinität zu der agilen Entwicklung nach Scrum und XP Praktiken (bspw. TDD, Continuous Delivery, **Pair-Programming**,
. fließend Englisch

...Stack: Java SE, JavaScript, CSS, HTML)
...Konzeption und Betrieb von verteilten Systemen
...Kenntnisse mit Apache Spark
...schenswert sind Erfahrungen mit NoSQL-Datenbanken und Responsive Design Webanwendungen
- Die Beherrschung der gängigen Entwicklungstools (IntelliJ / Eclipse), Jenkins, Git usw. setzen wir voraus.
- Nachgewiesene Projekterfahrung in der agilen Softwareentwicklung mit TDD/BDD, **Pair Programming**, Continuous Integration/Continuous Delivery und Scrum.
- Die Beherrschung der gängigen Entwicklungstools wie IDE, Jenkins, Git, usw.
- Gute Kenntnisse in REST und Testframeworks (bspw. TESTNG, Hamcrest, Mockito)
- Teamfähigkeit und gute kommunikative Fähigkeiten sind ein absolutes Muss.

Fazit – Pair Programming

- Es reicht nicht aus, "agil" zu sein
  - Agilität ist mehr als ein Scrum-Board und/oder JIRA
  - Agilität ist mehr als schnell ein Produkt abzuliefern
  - …

+ Pair Programming hilft dabei, agil zu werden *und zu bleiben.*

Fazit – Pair Programming Coaching (1)

+ Fördert *richtiges* Pair Programming

+ Vertieft TDD

+ Verbessert die Team-Kommunikation

+ Hilft, Team- und Qualitäts-Erosion vorzubeugen

- Kostet längere Coaching-Zeit, nicht nur einen Workshop

Fazit – Pair Programming Coaching (2)

- Erfordert Mut zur Entscheidung
  - Aber: Mut ist ein agiler Wert!
- Wer besser (agiler) ist als der Wettbewerb, gewinnt.
- *Pair Programming Coaching* lässt Programmierer ein wichtiges Hilfsmittel agiler Softwareentwicklung fundiert kennenlernen.
- *Einfach selber ausprobieren* 😀

Fragen?

31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank! 😊

Thomas Much

www.muchsoft.com
www.javabarista.de
twitter.com/thmuch