Pair Programming and the Re-appropriation of Individual Tools for Collaborative Programming

Sallyann Bryant
IDEAS laboratory
Dept of Informatics
University of Sussex
s.Bryant@sussex.ac.uk

Pablo Romero
IDEAS laboratory
Dept of Informatics
University of Sussex
pablor@sussex.ac.uk

Benedict du Boulay
IDEAS laboratory
Dept of Informatics
University of Sussex
b.du-boulay@sussex.ac.uk

ABSTRACT
Although pair programming is becoming more prevalent in software development, and a number of reports have been written about it [4] [6], few have addressed the manner in which pairing actually takes place [5]. Even fewer consider the methods employed to manage issues such as role change or the communication of complex issues. Here we contribute by highlighting the way resources designed for individuals are re-appropriated and augmented to facilitate pair collaboration

Categories and Subject Descriptors
Programming techniques, computing methodologies.

General Terms: Human Factors.

Keywords: Extreme programming, pair programming, collaborative software development, ethnography.

1. INTRODUCTION
Collaborative programming is not uncommon in the commercial world and has been formalised as 'pair programming' where "all production code is written with two people working at one machine" [1]. Two roles have been identified, the "driver", who is currently controlling the computer, and the "navigator", who contributes verbally (and subtly in other ways, as shown).

A number of studies have considered the costs and benefits of pair programming (e.g. [3] [8] [10] [11) however, none have closely considered how the roles of driver and navigator are dynamically realised and facilitated by the artifacts, environment and language used by the programming pair.

This paper draws on a detailed ethnographic account to highlight how pair programming is practically accomplished. It focuses on how tools are re-purposed and used alongside dialogue to facilitate role management and communication.

2. TEAMS OBSERVED
The data were collected from four, one-week studies of experienced pair programmers (with at least six month's continual commercial experience) in four companies. The studies took place in the workplace, with the programmers working on typical tasks in their usual environment. 36 one-hour sessions were observed, transcribed and analysed.

GROUP'05, November 6–9, 2005, Sanibel Island, Florida, USA.
ACM 1-59593-223-2/05/0011.

3. THE RE-APPROPRIATION AND AUGMENTATION OF SOLO ARTIFACTS

3.1 Keyboard
The keyboard consistently became a programming pair's primary token for 'floor control'. Possession of the keyboard avoided complications from having both programmers simultaneously editing the code. The keyboard was also often used to indicate intention of role change: the driver might slide the keyboard over to the navigator to suggest an exchange of roles. Although relinquishing control of the keyboard in this way was considered acceptable, initiating control of the keyboard was rarely seen.

3.2 Mouse
Although the driver would usually control the mouse it was not uncommon for the navigator to lean over and use it to 'point' at something on the screen. Presumably this was to avoid both the physical inconvenience of finger-pointing and the time and cognitive overhead associated with verbalisation.

3.3 Surrogate mouse
In one pairing session paperclips were used as an informal role control mechanism. When A was driving, B would take up the paperclips and make movements mirroring those A was making. When B wished to assume the role of driver he would let go of the paperclips and A would relinquish control of the mouse (and keyboard). Once finished as the driver, B then let go of the mouse and once more picked up the paperclips, at which point A almost immediately took up the driver role (and the mouse) once more.

Figure 1. The surrogate mouse

3.4 Interactive Development Environment
The code itself played an important role in communication and did not seem to be merely the drivers 'translation' of the collaborative effort or the 'product' under development. Sometimes talk would trail off and the interaction would be continued by typing at the keyboard. This was clearly the case where the navigator interjected

using agreement protocols normally reserved for conversations (e.g. “mmmn’ or ‘uh huh’). An example is:

> A: And so this is going to be…..(long pause while typing)….is this looking right so far?
>
> B: I think so.

The distributed cognition afforded by this representation often led to underspecified statements, as reported elsewhere [5]. An example is:

> A: Err…get this version of that….so that’s got that….so it’s come through there now.
>
> B: So if you try and run that through there now.
>
> A: Is this a problem?

3.5 Toys

On three of the projects seen, soft toys were used as tokens. A programming pair would collect the toy and place it on top of their terminal to indicate that they were currently loading new code onto the integration machine. Essentially these tokens were an informal ‘locking mechanism’. Their effectiveness relied entirely on members of the project understanding and conforming to their rules of use. This is particularly interesting as some other, more formal, technology based locking mechanism might just as easily have been put in place. It is also contrary to an example in Rogers and Ellis [12], showing that software developers were inconsistent in their use of a manual whiteboard for file locking as this was extraneous to the work activities they were involved in. In keeping with a number of studies in the field of CSCW (e.g. [7] [13]), the physical presence of the toy and its manipulation may alert others to peripheral events which might be of interest (here use of the integration machine). This is consistent with studies of news rooms, police operations, traffic control centres and operating theatres [7] in which participants were seen to “design and produce actions to render features of their conduct selectively available to others” . Robertson [13] stresses the human ability of peripheral awareness as particularly pertinent. In pair programming teams, each team member is given the opportunity to notice the change in integration machine control by the developer walking over and retrieving the toy. Even if this is not attained, the toy’s placement on top of the developer’s monitor makes it continually available to the rest of the team.

4. CONCLUSION

The re-appropriation and augmentation of solo tools suggests programming pairs have extra requirements from their workstations and environments. While this ‘re-purposing’ shows ingenuity and flexibility on the part of the programmers, it suggests that there is scope for the design of more specialised tools for use when pair programming in a collocated manner, providing specifically tailored tools for collocated collaborative software development rather than shoe-horning existing resources into collaborative use.

5. ACKNOWLEDGMENTS

This work was undertaken as part of DPhil research funded by the EPSRC. The authors would also like to thank the BBC iDTV project, BNP Paribas, EGG and LogicaCMG.

6. REFERENCES

[1] Beck, K. *Extreme programming explained: Embrace change*, Addison Wesley (2000).

[2] Beck, K., M. Beedle, et al. The Agile Manifesto. http://agilemanifesto.org (2001).

[3] Cockburn, A. and L. Williams, 'The costs and benefits of pair programming'. *Extreme programming examined*. G. Succi and M. Marchesi, Addison Wesley, 2001: 223-243.

[4] Dick, A. and B. Zarnett, 'Paired programming and personality traits'. *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering* (XP 2002), Alghero, Sardinia, Italy.

[5] Flor, N. and E. Hutchins, 'Analyzing distributed cognition in software teams'. *Empirical studies of programmers: Fourth workshop*, J. Koenemann-Belliveau, T. Moher and S. Robertson (eds). Ablex publishing corporation (1991): 36-64.

[6] Gallis, H., E. Arisholm and T. Dyba 'A transition from partner programming to pair programming - an Industrial Case Study'. Workshop: "*Pair programming installed*" at Object-oriented programming, systems, languages and applications (OOPSLA 2002) (Seattle, USA).

[7] Heath, C., M. Sanchez Svensson, D. Hindmarsh, P, Luff, D. von Lehn. 'Configuring awareness.' *Computer Supported Collaborative Work 11* (2002): 317-347.

[8] Heilberg, S., U. Puus, P. Salumaa and A. Seeb. 'Pair-programming effect on developers productivity'. *Fourth International conference on extreme programming and agile processes in software engineering (XP2003)*. Springer-Verlag, 2003: 215-224.

[9] Hutchins, E. *Cognition in the wild*. Cambridge, MA, The MIT Press (1995).

[10] Jensen, R. 'A pair programming experience.' *The Journal of Defensive Software Engineering* 16, 3 (2003): 22-24.

[11] Lui, K. and K. Chan. 'When does a pair outperform two individuals?' *Fourth international conference in Extreme Programming and Agile Processes in Software Engineering (XP2003)*. Springer-Verlag (2003): 225-233.

[12] Rogers, Y. and Ellis, J. 'Distributed cognition: an alternative framework for analysing and explaining collaborative working'. *Journal of Information technology* **9, 2 (1994)**: 119-128.

[13] Robertson, T. 'The public availability of actions and artefacts'. *Computer Supported Collaborative Work (CSCW 2002)* **11 (3-4)**, Kluwer Academic Publishers (2002): 299-316.

[14] Williams, L., R. Kessler, W. Cunningham, R. Jeffries. 'Strengthening the case for pair programming.' *IEEE software* **17**, 4 (2000): 19-25.