# Help! My... ...developer is running away

Oh dear. Your prize developer has been lured away to a better job in the Maldives - where the only thing it rains is money. In a moment of shock, you realise your developer is about to take a lot of critical software-development knowledge with him. Long before you receive your first Maldivian postcard, you want the remainder of your team to get up to speed with your prize developer's knowledge. What's the best way to approach this? The technical handover is key.

**This briefing paper is targeted at project managers whose project is about to lose an important developer. This paper provides guidance on how to perform a technical handover, which will ensure that the soon-to-leave developer will impart his or her valuable technical knowledge and identify traps and pitfalls you should avoid.**

It is important to perform a technical handover as early as possible - don't underestimate the time it will take for a sufficient handover to take place. Until your developer has left, their knowledge is a valuable asset. The transfer of software technical knowledge is often a time-consuming and complex task, so plan ahead, decide what's important, schedule regular meetings and slowly get your remaining developers to take over the leaving developer's tasks.

## Why is a technical handover important?

The software you have developed is a unique asset. It will often be a critical and integral part of your research, and represents a significant amount of effort. Similarly, your prized developer has invested a great deal of expertise and time into the software. This investment needs to be protected. Without a technical handover, the departure of a critical developer (to either to the Maldives or just another project) can leave your development team unable to do their work. Implement new features, bug fixes or maintenance could take significantly more time - and might not even be possible. A technical handover allows your remaining team to get up to speed on the leaving developer's work, leaving development unaffected.

It turns out that organising a good technical handover is rather like organsing an amicable divorce.

### Plan ahead to avoid unpleasantness

Have an initial meeting as early as possible with your developers to plan the handover. Decide on a prioritised list of technical aspects that need to be covered in the handover process, and how you're going to carry out the process.

### Divide up the mutual friends

Often, it's not just about knowledge: a developer will have many valuable contacts. It's important to make sure these contacts are introduced to the remaining developers.

### Give the unhappy couple enough time to sort things out

The leaving developer will have little time to tie up loose ends, so make it a priority to schedule regular and frequent meetings for the remaining developers to meet and absorb the right knowledge.

### Catch up every so often

Arrange to meet as a group to discuss handover progress on a regular basis to plug any knowledge gaps.

### Write up a divorce agreement

There is often value in recording the technical knowledge handover details. This document may save you considerable effort if you are expecting new developers to join the team in the future.

### Determine the marital assets

What are the aspects that need to be considered for handover? They may not be limited to the software, but may include related things as well, such as code repository, website and wiki maintenance, and documentation.

### Who gets the dog?

Decide which developer is best suited to inherit each aspect of your software from the leaving developer. You need to question which developer has the most appropriate skill set for each aspect. Is it best (and possible, given the time left) for the whole team to learn every aspect?

## Pitfalls to avoid

### Does only one of you know how to work the microwave?

Are there any key technologies that your remaining team need to learn about? For example, if your software is developed within a development framework (e.g. Grails, Ruby on Rails, Spring, etc), it will take time to learn the framework as well as the technology it is used to develop.

### Don't recycle old arguments

Of course, there's no need to cover aspects that your remaining team already know, or can be found out from existing materials. These materials might include tutorials, design documents, architectural overviews and the issue tracker.

### Don't just remember the bad times

Use this as an opportunity to extract the best ideas from your prized developer. They will have ideas on what needs to be improved, and how these improvements should be done.

## Further information and useful resources

**The STFC: an excellent synthesis of the topic and the Significant Properties of Software framework**

http://www.stfc.ac.uk

**The Software Sustainability Institute: resources, consultancy and collaborative projects for software preservation and sustainability**

http://www.software.ac.uk

**The Significant Properties of Software project - a JISC-funded study into what properties are needed to allow software to be systematically preserved**

http://bit.ly/eF7yNv

curtis+cartwright    Software Sustainability Institute    JISC

This briefing paper has beevn preprared as part of a JISC-funded project by the Software Sustainability Institute in partnership with Curtis+Cartwright Consulting Ltd. It is part of a series of outputs that have been devveloped to support the HE sector by raising awareness of software sustainability and preservation issues in the software-development process. Images courtesy of Vincepal (front) and billjacobus1 (back).