Improving the Success and Effectiveness of Software Developer Training

Ashley M. Aitken

School of Information Systems, Curtin University of Technology
ashley.aitken@cbs.curtin.edu.au

Running Code Productions
ashley.aitken@runningcode.com.au

Abstract

Keeping software developers up-to-date with the latest technologies, methods, and tools, and training new developers on the fundamentals of software development is an important task for managers of software developers and the software developers themselves. The planning and choice of what training courses to take, when to take them, and how to take them, is much more important than just "using up an annual training budget." This experience report explains what the author (a trainer with over seventeen years experience) has seen that does work and what doesn't work with regards to getting software developers trained (and keeping them trained). This report will suggest what training managers and software developers should do (and what they shouldn't do) to improve the success and effectiveness of their training and to get the most from their training budget. It considers, for example, just-in-time and just-enough training, intensive versus extended training, and many other aspects of instructor-led training (including the responsibilities of those attending the training before, during, and after the training sessions).

1. Introduction

Software development technologies, methods, and tools are changing at a rapid pace, a pace never before seen in the industry. As well, the amount of knowledge that developer needs to have, and the number of skills that they need to be proficient in, is now larger than ever before, and is growing even fast and changing even more every year.

In the past, it was often enough for developers to know a programming language (e.g. FORTRAN, or Pascal, or C) and have good problem solving skills. These days the language is often the smallest part of a developer's knowledge and skill base. Developers now usually need to know large development libraries and frameworks (e.g. Java SE or EE and Microsoft .NET), Web technologies, XML and associated technologies, development methods ranging from agile methods (like XP and SCRUM) to the more traditional methods (like RUP), and associated development technologies and skills (like unit testing, use-cases, OO analysis and design).

Of course, any developer worth their salary should be continually developing their own knowledge and skills, and any sensible organisation should be allowing developers time to do this (a half a day a week would be a minimum), to keep up with the changes in technologies, methods, and tools. This incremental approach is absolutely necessary, although not easy. Frequently, however, it is necessary to get a more significant upgrade in the knowledge and skills associated with a new technology, method or tool for development and this is where instructor-led training can assist much more than any of the other approaches.

The objectives of this experience report are to:

- Describe what the author has observed that works with regards to getting software developers trained so that managers and developers themselves can make their training

978-1-74067-550-5

more successful.

- Describe what the author has observed that does not work with regards to getting software developers trained so that training managers and developers themselves can avoid the same traps in the future.
- Share with training managers and software developers themselves some of the secrets of training that training organisations may not, generally speaking, want you know (since it may threaten their revenue stream).

A lot of the experiences and advice given within this experience report may seem somewhat obvious to the reader (at least it sometimes seems that way to me). That said, it amazes me how often I see these situations and contexts arise and how often clients seem not to learn from their past experiences. The latter probably being caused mostly by a lack of knowledge management within organisations (e.g. a training managers move on and the new one starts a fresh).

It is well known that the effectiveness of developers can vary by as much as twenty times (i.e. some developers can be as much as twenty times as effective w.r.t. productivity and/or quality as other developers). Of course, a lot of this comes down to the talent of the developer and their personal knowledge and skills. However, training can certainly assist those that aren't naturally talented (and even those that are) to develop software better, faster and cheaper.

In the next section I will give an overview of my training background and experience so that the reader can have some confidence in relation to the experiences presented, lessons learnt and suggestions made. Section 3 will discuss the value of instructor-led training (as opposed to other methods of learning) and Section 4 will discuss a number of experiences I have had in developer (and other technical) training and the lessons that we can all hopefully learn from these experiences. Finally, the experience report will finish in Section 5 with a discussion of a number of ways of improving the effectiveness and success of developer training.

2. Training background and experience

In my professional working life, I have been fortunate enough to be able to work in academia (as an academic doing both teaching and research) and in industry (as a consultant, developer, and professional trainer). I am grateful to my current primary employer, Curtin University of Technology, for allowing me this flexibility in my working life. Although often demanding, working in both these arenas has turned out to be very symbiotic and rewarding.

I have found that my industry work informs my teaching – students prefer it (rightfully so) when you teach from experience rather than just a textbook and my research can be more practically oriented – and my teaching and research (as a result of having time to go to conferences, read books, and reflect on material) informs my industry work (where everyone is usually too busy to read books, attend conferences, and reflect very much on what they do).

With regards to my industry training experience, I have:

- Been doing professional industry training since 1990, which means (as of 2008) I have approximately seventeen years experience in developer (and other technical) training,
- Given training courses ranging from an introductory (e.g. basic programming) to advanced level (e.g. OO design patterns and Enterprise development),
- Given training courses around Australia, across South-East Asia and in the US and UK to developers with a wide range of experience and cultural backgrounds,
- Given one-on-one training, training to small and large groups, and many train-the-trainer courses (wherein I train trainers who then go on to train others),
- Done a lot of training alone, i.e. solo training, but also worked with another trainer and a team of trainers,
- Provided training services for most of the large multi-national computer companies and many small, medium, and large organisations,
- Produced my own training materials for many courses but also used training material produced by a number of other organisations,

- Received recognition and awards for the quality of my training (and teaching) from Industry, student organisations, the University, and the Federal Government.

This section is not meant to be a CV or a sales pitch for my training services, but rather to hopefully demonstrate to the reader the extent and scope of my professional training (and educational) experience. This will hopefully give weight to the observations and lessons learnt that form the core of this experience report and the suggestions made with regards to making developer training more effective and more successful.

3. The value of instructor-led training

As most of the material that is presented in training courses can be found in textbooks and on the Web, one has to ask what extra value does instructor-led training have to offer? I believe the value of instructor-led training comes down (at the very least) to two things: 1) giving developers solid interruption free time to focus on learning new knowledge and practicing new skills, and 2) the motivation and enthusiasm that an instructor can provide. And I believe, the latter point should be by no means discounted or under-estimated. Even Olympic athletes need personal trainers / coaches to motivate them to get out of bed and train each day. Developers are the same; they need someone (or some course) to push them past procrastination and through the pain zone to successfully learn new technologies, methods and tools.

Instructor-led training is not cheap (particularly when compared to the free material available on the Internet) but it can (almost always) be guaranteed to be effective. Those taking training can usually rest assured that considerable effort and thought has gone into the development of the training material by those who develop it, and into understanding and working through the material in the training course by the trainer. For popular courses, you can generally rest assured that the material has been presented many times before and any issues in the material or its presentation have been well and truly worked out.

In 2005 I presented a poster on "ICT Training in the 21st Century" at the ICTWA Conference [1]. It discussed new approaches to training ICT staff that recognised: 1) a move to more individualized learning for individualized contexts, and 2) a move toward more flexible delivery (in terms of pace, duration, medium etc.) In particular it focused on new approaches to training that included webinars, podcasts, Web forums and mailing lists, Web tutorials, Web help, and Internet relay chat (or instant messaging). In particular, these approaches facilitated a much more incremental learning (in contrast to the more traditional and intensive instructor-led training discussed here). At the time these looked very much like the future of training.

In the last few years, however, training organisations have seen a move back to instructor-led training (at least there has been a large increase in demand for instructor-led training from the low levels of demand that were seen after the DotCom crash). Whether this is a result of the recovery of the ICT industry after the DotCom crash, or whether it is a sign that newer approaches to training (as outlined above) have not been as successful as anticipated, is not clear. What is clear, however, is that instructor-led training is not going to be fully replaced by these new approaches any time soon.

As mentioned earlier, often developers are so hard at work with their heads down to meet tight schedules that they don't have time to keep up with what is going on around them with regards to software development technologies, tools and methods. In particular, they don't have the time to acquire more significant chunks of knowledge and skills. As well, getting knowledge and skills via the grapevine of colleagues and / or the community on the Internet can sometimes be dangerous, propagating confused thinking, untruths, or just bad practices.

A classic example of the misunderstanding (that can result from "discussions" amongst developers) has to do with the understanding of "object-oriented frameworks" within the software development workforce. I believe a large number of developers (at least a lot that I have trained) have had a poor understanding of what object-oriented frameworks really are. Many confuse or conflate them with object-oriented libraries, probably because the terms are

978-1-74067-550-5

often used interchangeably by colleagues and in discussions on the Web and elsewhere.

Object-oriented frameworks, however, are very different from object-oriented class libraries, and have a precise technical definition. Resulting from this is the fact that the way object-oriented frameworks work and the way developers use them is vastly different than object-oriented libraries. To use the term incorrectly can lead to poor communication and poor software design. Many developers who have done my training have remarked how this clearer (and more correct) understanding of object-oriented frameworks has helped them greatly.

Professional trainers (and especially those that develop the training material) on the other hand have it as a part of their brief to keep up with what is happening with software development technologies, methods, and tools. For example, I generally spend at least one day a week (spread across the week) keeping up with the latest technologies, methods, and tools. This is done, primarily, by reading material from the Web, from books, and experimenting with new technologies, methods and tools, but also by attending academic and industry conferences.

4. Training software developers – experiences and lessons learnt

Before I discuss some experiences I have had with training software developers, and the lessons I hope we all can learn from these experience, I must say that these are offered as a way to improve software developer (and more generally any IT) training rather than being a criticism of past or present clients. As is often said, the customer is always right and I am grateful to all the organisations that have employed my training services.

As well, as mentioned earlier in this experience report, the aim here is to help clients get better quality and more cost effective training. Some (cynical folk) may that say that most training organisation are trying to maximize sales, and this would probably be true, but hopefully in most cases this is in the best sense of the word sales, i.e. meeting the customers needs, perhaps before they even know that they have a need.

4.1. Just-enough training never is

One of the catch-cries of modern training is *just-enough training*. This refers to the idea of giving software developers just the training that is absolutely necessary to get the job done (and as discussed in a later experience and lesson learnt, usually *just-in-time* to get the job done). Now there is nothing inherently wrong with this approach to training, no one should do more training than is necessary. The problem is that training managers and software developers often (usually?) under-estimate what they need to learn (or more correctly, what they already know, and what they need to know).

One good example of this was the training I did for many organisations in a major Web application framework (using material provided by the organisation that developed the framework). There were three separate and very high quality training courses provided by this organisation, specifically two one-week courses and a shorter three-day course. Unfortunately, a lot of the developers who attended the first training course were poorly prepared for the material (that was rather advanced and fast-paced) even if they met the stated pre-requisites.

To really get the most out of the first training course developers should have had previous training in the foundations of object orientation, object-oriented analysis and design (including topics on OO frameworks), object-oriented programming in general, the specific object-oriented programming language used. Unfortunately, although the pre-requisites for the training course generally made this clear, most developers who attended didn't have this background and as a result the training was not as effective and the results were no where near as good as they could have been (although I did everything I could to compensate for the situation).

Secondly, although a lot of developers were ultimately very happy (or at least intrigued) with the first training course – often as a result of the extra work I did to fill in the gaps in their knowledge – very few of them went on to take the second and third training courses. This was somewhat like learning to eat with a fork and knife but only taking the course in how to use a

fork. This framework had a very steep learning curve, but these courses represented a staged and well-thought-out and presented way to climb that curve. Leaving out the second (and possibly the third) training course was a recipe for ineffective development.

The lesson learnt from experiences like this is that *just-enough training never is enough. Training managers and software developers really need to consider closely (and honestly) what they need to know (and pre-requisites for that knowledge) to get the job done.* Of course, training organisations and trainers can often assist in this regard. Committing to training (not necessarily instructor-led training, but some form of training) in anything less that what is really needed is a sure path to failure (and a poor return on the investment in training).

4.2. Just-in-time training never is

Another of the catch-cries of modern training is *just-in-time training.* This refers to the approach of giving software developers training in what they need just before they need to apply it. Unfortunately, taking a training course doesn't make a developer instantly an expert in the technologies, methods, or tools. It usually just gives them a good foundation and good directions upon which they can build their knowledge and practice their skills in the particular area. To expect developers to come out of a training course and immediately be 1) productive and 2) capable, in what they learnt represents a significant misunderstanding of training.

In a number of instances I have seen organisations send developers on training courses expecting them to come out of the course highly productive and proficient (if not expert) in a new technology, new method, or new tool, so much so that they have expected a product to be developed and released (production quality) using this new technology, method, or tool, often within months of finishing the training. This just does not take into account how the human mind learns, particularly the time that it takes to synthesize new knowledge and to practice new skills. And this, of course, is one of the reasons such projects are often problematic.

The lesson learnt from experiences like this is that *just-in-time training never is enough. Training managers and developers need to really understand how long it takes to learn new knowledge and skills, particularly to a level of proficiency and productivity that can be depended upon to deliver.* I would go so far as to say that if an organisation believes their training is scheduled just-in-time then it probably means it is way too late. Combined with the problem of just-enough training discussed above, these two items, I believe represent the primary reasons for the failure of instructor-led training in meeting an organisations needs.

4.3. Just-down-the-corridor training has big negatives

Generally speaking, instructor-led training is usually done at quite a fast pace, and there is usually a considerable amount of material to cover and little time to cover it (once lunch breaks, tea breaks, and review sessions, for example, are removed from the forty hour week). To miss one or two hours can have dire consequences for the learning of an individual or the class. Unfortunately this is often what happens when training is conducted on-site (or *just-down-the-corridor* as it is sometimes referred to). Having ones office and ones colleagues close at hand can cause interruptions and represent a temptation to strong to resist.

On most (if not all) of the on-site training that I have done, one (or more) developers or managers get called out of the training room to handle crises or to attend meetings that couldn't be rescheduled. This leaves the trainer in a difficult position, i.e. carry on and "lose" someone, or adjust the training (take an early lunch or do some practical work) so that the individual will not fall behind. Similarly, some developers just can't resist the temptation to check their email or continue some work if the training machines have access to the corporate network. As mentioned earlier, I believe one of the main benefits of instructor-led training is getting solid uninterrupted time away from work, to stop the need to multi-task and to focus on learning.

Whilst on-site training does have its positives (e.g. less travel, flexible schedules) the lesson learnt from experiences like this is that *just-down-the-corridor training also has many negatives*

978-1-74067-550-5

and they usually outweigh the positives. I believe most developers actually like to get away from the office for training and it definitely improves the training and learning experience. In fact, I would go so far as to suggest that even for customised training for developers from just one organisation, it would be better to choose an off-site location for the training. If the organisation cannot afford to have all the developers off-site at any one time and for an extended period then it is better to split the class into two or more groups and extend the training over a number of weeks.

4.4. Pre-packaged courses are usually better than custom courses

When arranging training just for their own developers, organisations often ask for custom courses, that is where they (usually the training managers, senior developers or the developers themselves) choose what modules to include within the training. Sometimes this is done in consultation with the trainer but often organisations are adamant about what they want and what they don't want in the course.

The problem with such custom courses is that, like the experiences with *just-enough-training* discussed above, training managers and developers are sometimes not in the best position to evaluate what they need to know and what they *really* know about a particular technology, tool, or method. It's somewhat of a contradiction that developers who are going to take the training (presumably because they don't really know or understand the training material) feel they are in a good position to choose what is in the training. Even a senior developer who may assume they have the expertise and understanding to put together a custom course should make sure they really understand the capabilities of each developer and the content of each module before eliminating them from the course content just because, for example, they don't seem necessary.

The truth is that whilst a pre-packaged training course may not exactly fit every developer a lot of effort has usually been put into developing a coherent set of modules that fit together in a specified sequence. Removing one or more modules can leave holes and make the course less effective. And, if the training course doesn't really fit the majority of developers it is usually because they shouldn't be doing the course (they already have the knowledge and skills) or they should be doing another training course before this one (because they lack the pre-requisite knowledge and skills). Gross misjudgements of training are more significant and serious than taking an extra module that a few developers may already understand (but even then can usually learn a lot from when considering it as a review).

The lesson learnt from experiences like this is that *custom courses are often less effective than the pre-packaged course.* Training organisations often provide "training maps" that direct developers and training managers as to who should take which courses and which courses depend upon which other courses. That said, if it is thought within an organisation that a custom course may be more appropriate, it is imperative that the training organisation and preferably the trainer be involved in determining the composition of the training. And, be prepared to extend the amount of training required rather than reduce it – custom courses as a cost-saving measure are bound to fail in the long run.

4.5. Practical work as "homework" never gets done

Sometimes in custom training course or when training courses are running behind schedule the client will suggest (or request) that the practical component of the training course be done as "homework," i.e. during the period between training sessions in extended training courses or after the course in intensive training. On first consideration this may sound like a fair and economical approach – why pay a trainer to sit around whilst developers do the practical work? Unfortunately, it is not as simple as that and this approach most often leads to problems.

I have had a number of experiences with "practicals as homework" in both cases (intensive and extended courses). In most, if not all of the cases, the developers did not complete or even make a substantial effort at the assigned practical work. This is not to blame the developers,

978-1-74067-550-5 © 2008

they may not have been given adequate time to complete the work or encouraged to do so. The result, however, is that the developers effectively missed out on the practical component and I had to spend additional time in the training covering the salient points of the practical work (i.e. what the developers would have learnt in addition to practicing new skills).

The lesson learnt from experiences like this is that *practical work to be done outside of the training time does not work and makes the overall training less effective.* It is thus more effective to include the practical work in the training sessions. In reality, of course, organisations are not just paying for the trainer to sit around during these practicals, developers have the trainer at hand to ask questions, to work through issues, and discuss the solutions. Such discussions are by far more valuable than just providing the solutions for the practical work that may or will not be completed outside of the training time.

4.6. Training takes time, it's a reality of learning

The *just-in-time* experience and lesson learnt related to not having enough time between the training and the requirement for productive use of the knowledge and skills learnt therein. The *just-enough* experience and lesson learnt related to cutting back on the training content. This experience and the lesson learnt from it relates to the time allocated to the training itself. Many organisations seem to request more training in less time. They are always in a rush.

Unfortunately, learning is the one aspect of life that cannot be rushed. You can do it more effectively, you can do it more intensively, but you can't rush it. The human brain just doesn't work that way. Effective training takes time; time to introduce the concepts, go over the concepts, put the concepts in context, contemplate the issues, and to learn and practice the skills (more than once) and then to review everything.

I have had many experiences with organisations that wish to squeeze a weeklong training course into four (or less!) days. And this is apart from organisations that want to customise a course (i.e. leave out certain modules) or do practicals outside of the training time (as discussed, with less than stellar results). These are organisations that just want to do it all faster and quicker and, most importantly, cheaper.

The lesson learnt from experiences like this is *that training takes time, there is no rushing it, and any effort to do so will result in less effective and thus less cost effective training.* Don't cut back on the training time because the project is falling behind or the schedule is so tight. An inclination to do this should be a red flag that something else is significantly wrong (e.g. unreasonable schedules, poor planning, poorly trained developers).

4.7. Taking the right training is really important

It should be obvious that taking the right training course is really important. Unfortunately, this doesn't always happen, and usually for quite obvious reasons. Some times training managers are not technical enough (or have long been away from the technology) to understand which training courses are really appropriate, and the developers who have not done the training yet may not really understand what they really need (as opposed to what they think they need).

The classic example of this is organisations that have approached me asking for training in the Unified Modelling Language. In 99% of the cases what these organisations really needed for their developers was training in object-oriented (OO) analysis and design using the UML. UML is just a notation (a modelling language) for the artefacts of analysis and design and doesn't incorporate (i.e. is mostly independent of) any process or guidelines for doing OO analysis and design. To learn it on its own is usually a waste of time and money.

Similarly, I also see organisations that want training in a particular OO programming language rather than more generally in OO programming. Organisations should (and some do) use the most appropriate OOPL for the particular project, not the one that they did their last training course in. For a developer knowledgeable in OO programming, picking up a new OO programming language should be a task measured in (at most) weeks not months.

978-1-74067-550-5 © 2008

Taking this even further, I see organisations that focus their training on OO programming and an OO programming language but do not get any training in the OO libraries and frameworks that are used with that OOPL. For example, on the Java and MS.Net platforms there are considerably more important things to learn about the libraries (that consist of hundreds of classes) than there are to learn about the specifics of the programming language(s) used.

The lesson learnt from experiences like this is that *it is important to make sure that developers are getting the most appropriate training for their and their organisation's needs.* At the very least training managers should consult with their senior developers or training consultants (who should know the important distinctions, as above) not the junior developers who are generally not in the best position to make such a decision.

4.8. Theory and practice go together

Developers are often very practical people who wish to get down to work as soon as possible on any task. To match their personalities training courses are often very practically orientated, perhaps with a cook-book approach to practical tasks, and very little theory. Whilst, of course, this practical component is often the goal of a training course I believe that it cannot really be achieved without a good dose of theory (i.e. at the very least an explanation of concepts).

Most learning theories suggest that to really learn and understand practical material a student must have a good conceptual context within which to embed the skills. This doesn't need to be highly complex theory but should at least contain clear and simple definitions and descriptions of terms or diagrams giving an overview of how these terms and concepts relate and how practical work fits into the overall scheme of things.

A lot of the training material that I have used which was produced by other organisations has had exceptionally high quality practical components. However the same material has often lacked in its explanation of theoretical terms and concepts. One example, which relates to an earlier experience and lesson learnt, is the Web framework training courses that didn't explain and distinguish OO frameworks from OO class libraries. Without additional training (outside of the regular instruction) developers would have been left in the dark.

On the other hand, in my own training material and training courses I work hard to give clear and simple definitions and explanation of all theoretical concepts and to put the practical work into context by explaining the "bigger picture." The fact that his works has been confirmed by the very positive feedback I have received about the training material and the training courses themselves. Developers feel enlightened when they really understand what they are doing. Theory doesn't need to be difficult or incomprehensible.

The lesson learnt from experiences like this is that it is important *to look for training courses and training material that covers the theory as well as the practice involved in software development.* Doing so will provide developers with a much broader and more solid foundation and conceptual basis from which to apply and transfer their knowledge and practical skills. In the long run this is much more effective than just cook-book practical classes.

5. Suggestions for more effective and successful training

This section will present a number of suggestions for how to improve the effectiveness of your developer training. These are things that I, as a professional trainer, believe could help developers and their organisation get more out of their training. Not all will be appropriate for all organisations, but hopefully many of them will make sense in your context.

5.1. Motivate the developers to learn

Of course, developers should be keen to learn new knowledge and skills that will make them more valuable to an organisation (for all the obvious reasons). However, sometimes this is not the case, and some developers even see training as a distraction from development. Sometimes

978-1-74067-550-5

developers need a little motivation to make them take the training seriously.

This is, I believe, where general developer training can learn something from the "certification" style training courses run by some large computer companies (e.g. Microsoft). Having a small test at the end of training can give developers the little extra motivation to learn the material and thus considerably increase the effectiveness of the training.

I don't suggest that these tests be done on the last day of the training, but rather sometime in the subsequent week, to give developers some time to consolidate the learning. The tests aren't supposed to be mammoth or extensive, although they may contain written and practical questions, just something that (again) motivates developers to learn the material.

Getting a certificate for just attending a training course doesn't really account for much. However, getting a certificate for actually passing a test does at least demonstrate some learning. Most developers don't want to fail a test amongst their peers and, as well, this information can be useful for training managers and the developer's resume.

5.2. Make sure developers go in to the training with an open mind

Almost by definition, training involves teaching developers new concepts, new ways of doing things, and perhaps new tools. Training often involves an attempt to change the way we think, the way we work, and what tools we use. Like any change this can be threatening and difficult for most people (particularly for people set in their ways).

Although developers need, of course, to reflect on what is being taught to them, and not just accept it all blindly, there has to be a certain amount of trust and faith in the trainer, at least whilst they are undergoing training. It is important that any developers that go into training do so with an open mind – open to new concepts, new methods, and new tools.

After developers have absorbed the material and obtained some experience with the practices or tools contain therein then they can reflect more on its true value and applicability. What may have seemed outlandish, or impractical, or plain weird at first, can actually turn out to be a revolution in thinking and acting if you give the material enough time to demonstrate its value.

The classic example of this is the Personal Software Process (PSP) from the Software Engineering Institute at Carnegie Mellon University in Pittsburgh. It is a radically different way of developing, which adopts a truly engineering approach to software development, where developers literally time and record everything they do, develop a detailed personal process for development, and build a database of information characterising their development.

The very idea affronts a lot of developers when they are first exposed to it because this is not their idea of development. However, many of those who complete the training (with an open mind) and put the ideas into practice for a while, end up saying that it revolutionises the way they develop and cannot understand how they developed without it in the past.

5.3. Make the effort and take the time to match developers to the training

In more general education the practice of outcomes-based education is currently considered best practice. This is, basically, the idea that students only progress to new material when they have demonstrated the abilities and competency required for a certain level. Such an approach in developer training is probably also long over-due. Although a developer may attend a course this does not mean that they are competent in that material and ready to continue to the next level of training.

As a result, each developer in an organisation should be personally evaluated for the appropriateness – in terms of their level of ability and their role – of him or her attending any particular training course. An on-going audit of all developers' knowledge and skills can be most valuable in this regard. Instructor-led training is not cheap and just sending the whole team along to the same training course because it is the easy thing to do, is not always the most cost effective or sensible thing to do (for the organisation or the developer).

For example, in a team of 5-10 developers, it is more likely that there are a variety of skills,

978-1-74067-550-5

interests, and responsibilities amongst the developers. As a result it would probably be wise to send a few developers along to an analysis and design course, a few along to an advanced design patterns course, a few a long to a testing course, and a few along to a platform-specific or technology course (e.g. Java EE).

This would allow the developers to learn at an appropriate level and also to specialise their expertise (rather than all the developers having to be an expert in every aspect of development). There would no doubt, subsequently in the work place, be significant cross-fertilisation of the knowledge and skills within the team (especially if the other suggestions listed here are followed) making the training much more cost effective.

5.4. Spread the training over a number of weeks

The majority of instructor-led training happens in intensive mode, i.e. consecutive days of training (usually ranging from one to five days). This is an effective approach in that it is easy to schedule, completes the training in a short period, and doesn't require developers to context switch back and forth between work and training. However, it also has its problems. Many developers find a week (even a three or four consecutive days) of training too intense.

There is an alternative to intensive instructor-led training. Many independent trainers (and some training organisations) offer training over an extended period. Usually this means one or two days training for a few (usually consecutive) weeks. Of course, this approach also has some problems, e.g. that one must schedule resources and developers over a number of weeks, but it has significant advantages.

Extended training allows time for the material to sink in and for developers to contemplate, consciously or subconsciously, what has been discussed and practiced in the training within their own and organizational context. As discussed previously in this paper, this doesn't mean that developers can be counted on to do any homework outside of the training (so don't arrange for any) but it does seem to make the training more effective and less demanding on developers.

Running an intensive training course is also demanding for the trainer and so extending the training over a number of weeks can mean the trainer is fresher and has more time to prepare for each training day. This approach also allows developers to continue with their assigned work, as they don't have to be away from their job for as many consecutive days. Generally, speaking developers (and trainers) seem to enjoy training more when it is spread over a number of weeks.

5.5. Pay for developer mentoring and "booster shots"

As discussed earlier, developer training should not be seen as a once-a-year obligation of the employer. It should be seen as a necessary and useful means of keeping developers up-to-date with changing technologies, methods and tools. As such, don't just stop with one week of training a year. Consider having the trainer take on a mentoring role within the organisation or, at the very, least coming back to give some "booster shots" at a later date.

Mentoring is a process where the trainer comes back to the organisation and works with the developers (usually those who attended the training but perhaps a larger group) on issues that have arisen after, but usually as a result of, the training, presents updates and changes since the training occurred, and perhaps goes into more depth on particular areas of the training (as requested by the developers). This can be done once a month or more frequently.

"Booster shots" involve less regular visits by the trainer, say once or twice a year, just to progress the training, ascertain what the developers have been up to and what issues they have faced, recommend any further reading or training and answer any general questions the developers may have. Organisations should not see training as a one-off involvement with the trainer (or training organisation) but as the start of an on-going professional business relationship.

Trainers and training organisations are in a good position – since they keep up with changes within the industry and across organisations – to assist with the planning and direction of

training within an organisation. A trainer can work closely with a training manager to ensure that the training chosen is the most appropriate and cost effective so that developers will more likely be ready to perform productively on a project when needed.

5.6. Provide post training consolidation time

Often developers don't finish all of the practical work (or even digesting all of the theory) within a training course. As a result, developers should be given time, subsequent to the training, to complete the training and to consolidate the learning. This could range from one day to a few days, intensively or as the developers continue their regular duties.

Importantly, it is not what the trainer does in the training room that really solidifies any learning and skills in the developers; it is what the developers do themselves. It's whether (or not) they choose to, and have the time to, put together (within their head) all the pieces of new knowledge, to really practice the new methods, and / or to get familiar with the new tools.

This post training consolidation can be done individually or as a group (i.e. the group of developers that attended the training). It should also be seen as a requirement to have completed the training, not just an optional activity. Schedule this time and a post consolidation discussion with developers to ensure that it is done and to evaluate the training.

5.7. The best way to guarantee learning is to encourage them to teach others

One of the first things one learns as a trainer (and an educator) is that having to teach some material really motivates you to learn it well. Having to present a topic successfully to others mandates (or at least should require) that you have a solid understanding of the material yourself. Thus, one way to ensure that (at least some) developers really learn a topic is to get them to teach it to other developers in the organisation.

In this approach rather than sending the whole team or a large group of developers along to a training course, the organisation sends just a few developers (usually at least two). These developers are then required to eventually (after a consolidation and preparation period) present the training material to the rest of the team or organisation by presenting the course (pretty much as a regular training course) in-house one or more times to other developers.

This is not to say that these developers need to be first-class presenters or trainers but they should at least be able to explain the material to others in the organisation. Such an approach also makes the training more of a class exercise because these trainers aren't expected to be "oracles" with regards all aspects of the material. That said, this approach often leads to these developers becoming the in-house experts within the organisation with regards to this material.

5.8. Make sure the training includes a good balance of theory and practice

As discussed in the previous section, practice without theory can be as bad as theory without practice. It is thus important that any training has a good balance of theory and practice – use this as a criterion for evaluating training course or in discussion with the trainer or training organisation if putting together a custom training course.

Some developers may express a dislike for theory or, even worse suggest they don't need it. It's not possible to work in practice without a conceptual model, whether it has been explicitly developed or not. Giving developers free-range to construct their own conceptual model(s) leads to problems (with models differing amongst developers and with problematic models).

It is also important to make sure the practical sessions are long enough (and involved enough) to really put the theory into practice. And make sure that these sessions involve clear tasks and either a discussion of the solutions or issues that are raised at the end of the practical or provide solutions to the practical work.

978-1-74067-550-5

5.9. Finding the right trainer is crucial to success of a training course

Finding the right trainer can be a difficult task. Often with training organisations, the client will not be aware of who the trainer is until the training commences at which time there is not much that can be done. In every case an organisation should ascertain before signing up developers for training who the trainer will be, and if the trainer is not well known to them ask for a copy of the relevant portions of his or her resume.

Be aware also of the practice of "bait and switching" with regards to trainers. Some training organisations use a well-known trainer in advertising but they may only present the first session on the first morning and then hand over their training to their colleague. This is not always detrimental as sometimes the colleague may be more up-to-date with the material than the well-known trainer who may now be more involved in management instead of technical activities.

If you have a chance to communicate with the trainer before the training course make sure you ask them the following questions: 1) how many times have they presented this training course, and 2) when and where was the last time they put the theory and practice covered in the training course into real-world practice. If this is the first time the trainer has presented the course and/or they have not had recent real-world experience with the material then be careful.

5.10. (At the very least) standardise terminology across the organisation

One of the most difficult but also most important tasks within any organisation (and life in general) is communication. Poor communication usually leads to poor quality and reduced effectiveness within an organisation. One of the best ways to improve communication is to use clearly defined (and shared) terminology and, if possible, conceptual frameworks.

I have been asked by a number of organisations to come back to (briefly) train other staff (including non-development staff, e.g. business analysts) within the organisation in the basic terminology and conceptual frameworks I have presented to their developers This is a suggestion I now make to all organisations considering developer training.

6. Conclusion

This experience report has discussed a number of experiences I have had over more than seventeen years giving developer and other technical training. It has noted a number of lessons learnt, lessons that I hope will help training managers and developers not make the same mistakes others have made. The experience report has also presented a number of suggestions on how organisations can make their developer training more effective and successful.

A lot (but not all) of these experiences, lessons learnt, and suggestions would seem to imply that organisations should be spending much more time and energy on training, and, yes, more money on training. This may sound self-serving from a trainer who provides training services but I trust the experiences and reasoning given indicate the truth of the matter. And if the reader is still not convinced, consider this quote from Arie DeGeus of Royal Dutch / Shell:

> "The only competitive advantage the company of the future will have is its ability to learn faster than its competitors."

If your organisation doesn't really understand that spending more money on training and other learning-focused activities will be money very well spent (i.e. one of the best investments an organisation can make) then it would seem your organisation may not do well in the future. Investing in developer training (and other developer-related concerns) is the best way to grow and keep the best developers and we all want to do that, don't we?!

7. References

[1] Aitken, A.M. "ICT Training in the 21st Century", ICTWA Conference. Perth Australia, 11 November 2005.