Class 345

How to Get the Training you Need

Niall Cooling BSc CEng

Feabhas Ltd
5 Lowesden Works
Lambourn Woodlands
Hungerford
Berks RG17 7RY
UK
Tel. +44 1488 73050
Fax. +44 1488 73051
Web: www.feabhas.com

Abstract

Many engineers in the embedded field struggle to get funding for training. The root of this problem is that there are no clear requirements of what skills an embedded programmer should have. This paper explores why this is the case, and looks at how, when looking for training, an engineer can help themselves by developing a value model. This models ties training needs to project benefits, and builds a business case (which is typically uncomfortable for the engineer to do).

How to Get the Training you Need

Let's start with this paper's credo[1]:

> *Well-run commercial companies spend money for one of two reasons:*
>
> - *to make more money*
> - *to reduce overheads*

Motivation

Over many years of delivering technical training to the embedded systems sector, two questions really began to bother me:

- Why, when the economy get difficult, are training budgets one of the first things to get cut?
- Why do students, who have waited months to attend training, get pulled off mid-course?

Even though a myriad of articles tell us why businesses should invest in training during an economical downturn (1) funding for training is very hard to come by. So why does this happen?

In reality there are only two types of training:

- Must-do
- Everything else

Must-do training is unavoidable or compulsory training that a company must provide for its staff. *Must-do* training often has some legal requirements and typically has some form of test or certification of attendance. The most common example is First-Aid at work; in the UK Employers have an obligation under the Health and Safety to make adequate and appropriate First Aid Provision for their workforce (2). If a company doesn't do this then they are liable for prosecution. Another example often considered *must-do* is company induction training. The consequence of not running induction training is the potential of grievance tribunals for unfair dismissal etc.

Ultimately you cannot run a business without *must-do* training. The upside is that this type of training does not get cut and it is rare to get people pulled off this mid-course. In addition, *Must-do* does not require justification such as Return-on-Investment (ROI) calculations. Nevertheless, cost is seen as a major factor, so companies often attempt to do this type of training as cheaply as possible, to as many people as possible (as shown by the growth of e-Learning in this area).

[1] A Latin word which means "a set of fundamental beliefs or a guiding principle"

That then leaves us with *everything-else* training From a business perspective there is no compelling regulatory or legal reason to perform this training; it is therefore considered non-essential. Typically this training is *improvement* based training.

So which category does training related to embedded software development sit in? Quite clearly there are no legal requirements to be able to develop software[2]. All technical training is therefore "not essential" and falls under the *improvement* training umbrella. Unfortunately, many managers (and indeed, companies) see this type of training only as an overhead, which they *have* to do to keep people happy. Their aim in providing training is to reduce staff turnover and absenteeism whilst keeping cost to a minimumThe classic giveaway is any company which allocates $X / engineer / year for training.

The root problem here is that training is seen only as a *cost*. No attempt is made to understand its true *value*. Non-essential training is therefore easy to cut funding for and easy to justify pulling people off due to “project pressures”. So how can we attempt to redress this?

Given that most technical training isn’t classed as business critical, let’s investigate why we do *improvement* training in the first place.

Clearly we have a job to perform: developing embedded systems. This is a collection of tasks that we have to perform; ultimately unsupervised. To perform these tasks we require a set of skills and knowledge appropriate to the task in hand. Thus we can define someone with the appropriate skills and knowledge to take on a task unsupervised as being considered *competent*.

On-the-Job Training

Unfortunately most people entering the embedded software industry (including graduates) today don’t have the core competencies to develop software for embedded systems unsupervised. Typically knowledge and skills are developed through the euphemism of *on-the-job (OnTJ) training*. This ‘training’ typically entails little more than examining existing code, asking questions and modifying or building-on this existing codebase. *OnTJ training* has been shown to have many problems, most notably (3):

- It is inconsistent – typically no goal is defined
- It is inefficient – the hidden cost associated with checking the work
- It is ineffective – engineers inherit poor practices which are then passed on

This leads to engineers developing live projects with a lack of foundation knowledge and unfortunately their mistakes often go into business-critical software.

Off-the-Job Training

Herein, companies may look to develop competency through “off-the-job” training (typically through a third party training company) with an implicit justification that post-course the engineer will be more productive, make fewer mistakes, ultimately leading to less testing and rework.

[2] This opens up a whole can of worms regarding the term “software engineering”; but that is outside the scope of this paper

Hopefully this is all pretty straightforward. Unfortunately here in lays one of our root problems. To engineers, developing and extending their skills through trainingis a “no-brainer”;yet we still see this type of training being cut. The problem is that we haven’t yet move from a cost-model to a value-model. Even though everyone in the engineering department knows this is a good idea, the accountancy team may believe that attending *Stress Management Training* is essential to their function and appropriates the training budget[3].

Central to moving to a value-model is understanding the concept of a *value-chain*. Remember our credo:

> *Well run commercial companies spend money for one of two reasons:*
>
> - *to make more money*
> - *to reduce overheads*

At the bottom end of our chain is the cost and expense of a commercial technical training course, at the other is our credo. Ultimately we have to link one to another.

Training maturity

Luckily for us, much of the hard work has already been done. Probably the best known[4] and most widely used approach for value analysis is that of Donald Kirkpatrick’s 4-level evaluation model (4) . This was originally developed back in 1959 and gained popularity in the 1970’s. The four levels can be summarised thus:

- Level 1 Evaluation-Reaction
  - This is when most training is at today. This typically involves attendees of the training filling out post-course questionnaires (*happy-sheets*). Naturally this form of feedback is highly subjective[5].
- Level 2 Evaluation-Learning
  - The second level aim is to be objective by attempting to measure retained knowledge normally through some form of post-course assessment/test. This is popular with Microsoft based courses (e.g. MCSE).
- Level 3 Evaluation-Behaviour
  - The ability to pass a test or demonstrate a skill in a classroom setting is not the same as being able to do so in workplace conditions, so how is the learning being applied at work? For example can we demonstrate that as a consequence of training an engineer is more productive or that the team are generating fewer bugs?
- Level 4 Evaluation-Results
  - Level four attempts to answer the ultimate question of *what value did the training add*? Here we address business issues, for example where are we spending our time /

[3] And yes, this has happened again recently
[4] But surprisingly still not that well known
[5] I have had feedback from courses wondering if the people were actually on the same course

money? If in maintenance, then what is the cost of fixing bugs, and has the training reduced that cost?

Armed with the Kirkpatrick model we can apply this framework to help build a business case for getting funding and approval for technical training. The approach and business case depends on whether you are looking at "here and now" (tactical) training or long term career development (strategic) training.

Regardless of the training need, there are two other pieces of information you'll need to establish before putting together the business case; these are:

- The key stakeholders
- The "yes" person

Identify the stakeholders

A stakeholder is someone (or group of people) who can affect the business case; usually benefit from it; but may also constrain it. First we have direct stakeholders: the proposed training attendees. It is pretty obvious that these are direct beneficiaries. In our case, however, we are interested in those stakeholders who have influence over whether the training will be approved. For each stakeholder it is important, where possible, to understand what their motivations and problem areas are. Aligning a business case to these specific needs significantly improves the probability of getting approval and sign-off. Typical key business stakeholders in an engineering organisation are:

- Business Leadership Team Member (CEO / COO / CTO / etc.)
    - Key strategic issues:
        - Reducing costs
        - Growing revenue and market share
        - Growing profit
        - Operational excellence/improvement
    - Possibility of 'pet projects' / interest in:
        - Quality improvement
        - Cost saving
        - Increased speed to market
- HR Directors
    - Key strategic issues:
        - Talent shortage in industry
        - Talent attraction
        - Talent retention
    - Interested in:
        - Being innovative
        - Showing Return-On-Investment (ROI)
        - Building a strong talent pipeline
        - Making a strategic contribution to the business

- Engineering Managers
    - Key issues:
        - Safety
        - Managing quality
        - Brand
        - Late shipments (bugs, rework)
        - Cost reduction
        - Reliability/quality
        - Time to market
    - Interested in:
        - Improving quality (and decreasing the massive headache of sub standard quality e.g. product recalls)
        - 'Eeking' more value out of the Time-Quality-Money triangle
        - Improving speed to market

The Business Leadership Team member is usually the most difficult to identify or get access too. Reading the internal company newsletter is always a good source, also look for who in the company is blogging or using twitter.

Find the *Yes* person

The second and quite often most important piece of information is identifying the *Yes* person. This concept was introduced to me many years ago by a friend who happens to be a very good sales and marketing specialist (5). Depending on the size of the company there is a purchasing chain; made up of a series of people who have to sign-off and agree to the training.

This chain is actually composed on a number of *No* people with a single Yes person (who is not necessarily at the end of the chain). Even though we might think of our boss being a *Yes* person, unless they actually control the budget they are classed as a *No* person. This means that a "no" from them will kill the proposal, but a "yes" doesn't actually mean "yes the training can go ahead", it only allows us to progress to the next link in the chain.

The *Yes* person is that key person that when they say "yes" it actually means "Yes the training will go ahead". It is very likely that the *Yes* person is also one of your key stakeholders.

Building the business case for "Here and Now" training

"Here and now" training is about addressing a very specific knowledge shortfall. The training is often very obvious to you, immediate, absolutely necessary and quite often project-critical The ideal strategy here is to position the training as "must do", but this is quite tricky. This option is best where you can align the training with the Business Leadership Team Member's pet project. A huge benefit to you is that "must do" does not require trying to define the ROI, but means it needs couching in the form "if we

don't do this the consequences are...". For a must-do those consequences need to be either project failure or even company failure.

For example, if a project has forced upon it a change of Operating System (OS), then you can probably build the case that unless the engineers receive training in that OS within a given timeframe then the project deadlines cannot be met, incurring late shipment, extra cost, unhappy customers, etc. There will be a real business cost associated with this – they key is being able to identify that cost.

This approach has two problems. First, having the confidence that if the training is not approved then the project will fail, as this proposition may reflect badly on you, your manager and all the project team (e.g. you are classed as someone who can't rise to the "challenge"). Second is the danger of "calling wolf" (6), in that this approach make work once, but is unlikely to work twice.

The alternative to the "must do" approach is building a business case as "added value" training. This option tends to be best suited to HR directors. Using the Kirkpatrick model as the framework we must establish a value chain. Remember our credo:

> *Well run commercial companies spend money for one of two reasons:*
>
> - *to make more money*
> - *to reduce overheads*

To build the value chain we have to look to establish expenditure to added-value. So, the target is to provide answers to either (or both) of the following questions:

- How will this training make the company more money (or market share)?
- How will this training save the company money?

Ultimately this will require an answer defined in terms of ROI, where ROI is simply:

> *((Gross benefit from training ($) - Cost of training ($)) / Cost of training ($)) x 100(%)*

But before we go any further the bad news is that building the value chain is near-on impossible in most organisations. Using Kirkpatrick, moving to each level has a greater and greater problem:

- Level 1 – level 2: Most training courses don't have any form of assessment, and those that do tend to have very superficial multi-choice tests.
- Level 2 – level 3: This is where there is a major chasm. To establish level 3 we need some quantitative definition of improvement and a consequence of the training. Probably bug count reduction is your best hope, maybe less time in testing or more lines-of-code per engineer per day? In my experience it is so rare to find useful metrics being collected and analysed. Sometimes companies attempt this with broad questions such as: "*Please estimate how much you expect your job performance related to the course subject matter to improve as a result of this training and other business improvements in your organization?*"

Level 3 – level 4: This is almost impossible to measure. Companies typically understand the *consumption* (costs) of each department but very few companies measure how each department adds *value* (in $) to the company; let alone each engineer! So truly getting to level 4 in most organisations is a nirvana. If a company is truly operat*ing at CMMi* level 5 (7) then, in theory, you should be able to build *a va*lue chain to level 4 (and much of this should have been done as part of the CMMi development). But I've yet to see it (even among CMMi-5 companies).

So do we just give up and go home? Of course not, but we have to start thinking and talking like the Marketing/Sales people. We need to be able to "sell" our proposition.

As an example, we have a case study where the value chain was positioned as in terms of business cost. The HR director of our clienthad two immediate problems:

1. The mechanical engineering aspects of the company had just been outsourced. This left him with 6 mechanical engineers who had valuable domain knowledge but no work.
2. There was a shortage of good embedded software engineers (ESWE) and they were having difficultly recruiting.

There was clear $ cost associated with making these people redundant and this also did not address the second problem of a lack of ESWEs.

Using our FastTrack Graduate Training programme (8) as a basis, we could demonstrate that if the training could reduce the timeframe for the engineers becoming competent from 2 years to 18 months this would result in a potential saving $300,000. An assessment model was used to move from level 1 to level 2, which both objectively (via an exam) and subjectively (via instructor feedback) demonstrated the gained skills and knowledge of the students. The reality was that after 9-weeks of training certain students had progressed so well that they were considered competent to contribute to the business immediately.

Long term career development training

"Here and now" training is all very well, but the problem is that it can be like waiting for a bus, you can wait for ages then suddenly a number come at once. Instead, we need to take a longer term view and do something that most engineers don't do, which is *drive* the appraisal process.

All companies (should) go through an annual appraisal process. Unfortunately in most cases these tend to be very general (after all the appraisal form is usually the same form for everyone in the company) and a bit "woolly".

The approach here is to build your own definition of a "competency" model for ESWE. This then enables training-needs-analysis (TNA) / skills-gap-analysis (SGA) to identify a training plan of action. The aim is to build a compelling argument targeting the issues and interests of both the Engineering and HR directors.

Focusing on competency is directly addressing the engineer manager's key interest in quality. Then there are a series of steps:

1. We need to get the Engineering Manager to agree that quality is central to the business goals (if your engineering manager doesn't agree with that then I'd start looking for another job).
2. We the need to define the competencies required by engineers to attain this quality (the competency model)
3. We need to assess where people are today against this model (skills-gap-analysis)
4. We need to plan a structured approach to attaining this competency (training-needs-analysis).

The mistake regularly made is that competency is seen as a pass/fail assessment, rather than a structure based and skill and knowledge. A competency model has four layers to its structure:

Layer 1 - Competency Areas

Within any project there are many different competencies required, for example project management, QA, technical authoring, etc. However, here we are limiting it to embedded software engineering.

ESWE differs from traditional software engineering in that it has to not only has to encompass most of the normal software engineering aspects, but also has to deal with both systems and hardware subject areas (Figure 1).

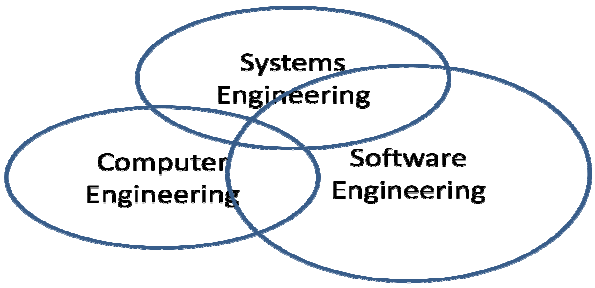

Figure 1 Embedded Software Engineering

These additional domains (computer and systems engineering) significantly affect the development of embedded software. These *Design Forces* are, for example:

- Performance
- Resource constraints
- Cost
- Reliability
- Power management

Layer 2- Competencies

Within a competency area, we can broadly define the core set of knowledge and skills generally accepted to perform this job function. This, of course, will always be an area of debate as no two companies and no two jobs are ever equivalent. Nevertheless, by looking at accepted practices and teaching from academia and industry we can define a common group. As a good starting point the IEEE published the *Guide to Software Engineering Body of Knowledge* (SWEBOK). The objectives of the SWEBOK project are to (9):

- characterize the contents of the Software Engineering Body of Knowledge;
- provide a topical access to the Software Engineering Body of Knowledge;
- promote a consistent view of software engineering worldwide;
- clarify the place of, and set the boundary of, software engineering with respect to other disciplines such as computer science, project management, computer engineering and mathematics;
- provide a foundation for curriculum development and individual certification and licensing material

SWEBOK organises the material into Knowledge Areas (KA), with the core five being:

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance

The core competencies are the same for embedded software development as they are non-embedded software.

Layer 3 - Knowledge and skills for each competency

Using SWEBOK as a guide, each competency for software engineering can be further refined into a key set of skills and knowledge.

For example, the Software Design competency can be refined as:

- fundamentals
    - General Design Concepts - In the general sense, we can view design as a form of problem-solving.
    - Context of Software Design - it is important to understand the major characteristics of software requirements analysis vs. software design vs. software construction vs. software testing
    - Software Design Process
        - Architectural design
        - Detailed design
        - Enabling Techniques
        - Abstraction
        - Coupling and cohesion
        - Decomposition and modularization
        - Encapsulation/information hiding
        - Separation of interface and implementation
        - Sufficiency, completeness and primitiveness
- Key Issues in Software Design

    - concurrency
    - control and event handling
    - distribution of components
    - error and exception handling, and fault tolerance
    - interaction and presentation
    - data persistence
- Software Structure and Architecture
    - Architectural Structures and Viewpoints
    - Design Patterns (microarchitectural patterns)
    - Families of Programs and Frameworks
- Software Design Quality Analysis
    - Quality
    - Quality Analysis and Evaluation
- Software Design Notations
    - Structural - static view
    - Behavioural - dynamic view
- Software Design Strategies and Methods
    - function-oriented
    - object-oriented
    - data-structured
    - component-based

Software Requirements

The core competency for software requirements is not required by most embedded software engineers. Nevertheless, those engineers in smaller teams, or who are heavily involved in the system's definition will need to understand, and may have to apply, requirements analysis techniques such as domain specific languages (DSL), formal method notations, use cases, or general purpose languages such as SysML

Software Design

Software Design techniques is, undoubtedly, dominated by *UML*. Underlying this is, of course, is *Object-Oriented design*, and *Design patterns*.

Modern software development processes are all driven by the concept of *Incremental & iterative* design. Well known among these is the *Rational Unified Process* (RUP), (seen by many as too heavyweight for many smaller commercial projects). Alternative philosophies are:

- Agile – is suitable for embedded systems?
- Test Driven Development
- Model Driven Design

The design forces of embedded software are often different to those of other software systems. Typically, there is a much higher emphasis on timing and concurrency in the design. When using notations like UML language constructs like the following take on a much higher precedence:

- Active and passive objects

- Synchronous and asynchronous messages
- State models
- Sequence diagrams
- Timing diagrams
- Translation to code
- C / C++
- Real-Time Operating Systems

What mustn't be ignored are the function-oriented strategies. Currently these are mostly proprietary by nature, with two leading products:

- MathWorks' Matlab
- National Instruments' Labview

Software Construction

C is still the dominant programming language in embedded systems, followed by C++. There are many other programming languages of note (Ada, Java, Forth, etc.), but C is certainly the most important.

As with any programming language an engineer must have a strong grasp of the core language. However, when programming in C for embedded an engineer must be competent with areas such as:

- General Purpose Input/Output (GPIO)
- Peripheral register access
- Interrupts
- Memory layout
- Compilation/Linkage model
- Performance optimizations
- Architecture specific aspects, e.g. ARM-ABI
- Safe coding standards (e.g. MISRA-C)

A key issue in embedded software design is concurrency. This requires good understanding of multi-tasking/threading issues. The majority of embedded systems use proprietary RTOSs (e.g. VxWorks) however there is a growth in *Platform* based systems (e.g. Linux and pThreads).

Finally, it is become less and less common for embedded systems to be stand-alone. Therefore knowledge of networking and the skills to implement distributed systems using technology such as TCP/IP, CANbus and USB is becoming central to embedded software competency.

Software Testing

Many of the core topics within embedded software testing do not significantly differ from non-embedded software testing. What makes life different, and thus the skills and knowledge are

- The type of problem to test (e.g. timing and memory exhaustion)
- The tools available to test an embedded system (Figure 2)

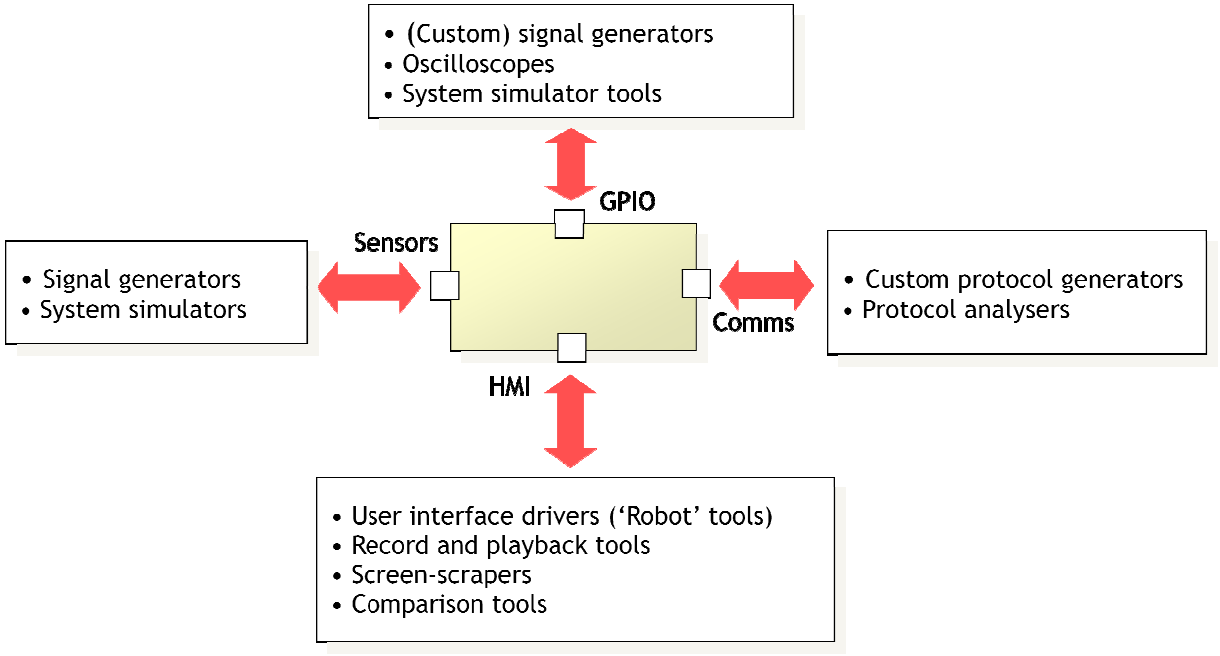

Figure 2 Potential tool requirements for testing an embedded system

Common tools include Logic analysers, In-Circuit Emulators, In-Circuit Debuggers from companies such as Lauterbach and iSystems.

Software Maintenance

Similar to software testing, many of the core topics within embedded software maintenance do not significantly differ from non-embedded software maintenance. The key topics are:

- Change Management
- Revision Control
- Build Process
- Release Process

Layer 4 - Competence at level 1..n

Where many certification schemes go astray is that they are based on a pass/fail model. This is inappropriate as people's skills and knowledge are constantly evolving. What is required are a set of competency levels based around autonomy of work (Has the person the authority and responsibility for all aspects of a significant area of work, including policy information?). At the lowest level, typically new graduates, they only need skills and knowledge to follow instructions (*ability to apply*), whereas a very experienced engineer has to reason and make personal choices (*ability to conceive*).

Industrial Examples are typically broad, e.g.

- supervised practitioner
- practitioner
- expert

Whereas companies have job/role specific levels, e.g.:

- Graduate
- Engineer
- Senior Engineer
- Principle engineer

Combining layers 2, 3 and 4 gives a 3-dimensional Competency Matrix (Figure 3). The axes being:

- Competency Levels
- Competencies
- Competency skills/knowledge

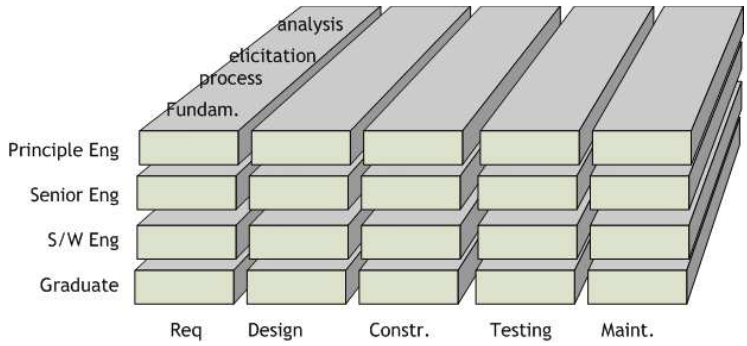

Figure 3 Competency Matrix

This now defines a full competency model. The intent of using the competency model is to get management buy-in. Once established, the competency model should become central to appraisals and career development. It gives you something concrete to plan for.

Conclusion

So, are any of the techniques presented guaranteed to get you the training you want? Of course not. Nevertheless, the intent is to show that unless you start approaching your own career development in the context of a business case, you will always be open to our underlying issues of lack of funding and sporadic training.

So, in review, what is our approach?

- Know your stakeholders
- Who is the “Yes” person?
- Identify and what type of training your after?
    - must do
        - If we don't do this the consequences are...
    - added value
        - How?

        - make more money
        - reduce costs / save money
    - quality model
        - develop a competency model (SWEBOK)

Always come back to our Credo: well run companies spend money for one of two reasons. So, take responsibility, build a business case and have specific goals. Good luck.

Bibliography

1. **Lumley, David.** *Revolution Learning and Development.* [Online] http://www.revolutionlearning.net/.

2. Publications. *Health and Safety Executive.* [Online] http://www.hse.gov.uk/pubns/indg214.pdf.

3. **Sissons, Gary R.** *Hands-On Training: A Simple and Effective Method for On-the-Job Training.* s.l. : Berrett-Koehler , 2001.

4. **Kirkpatrick, Donald and Kirkpatrick, James.** *Evaluating Training Programs: The Four Levels.* s.l. : Berrett-Koehler. 1576753484.

5. Simon West. *LinkedIn.* [Online] http://www.linkedin.com/in/simonawest.

6. The Shepherd's Boy and the Wolf . *Aesop's Fables.* [Online] The Literature Page. http://www.literaturepage.com/read/aesopsfables-86.html.

7. Welcome to the CMMI® Website. *SEI.* [Online] CMU. http://www.sei.cmu.edu/cmmi/.

8. FastTrack Graduate Training Programme. *Feahbas Limited.* [Online] http://www.feabhas.com/fasttrack.html.

9. **IEEE.** Guide to the Software Engineering Body of Knowledge. *IEEE Computer Society.* [Online] http://www2.computer.org/portal/web/swebok.