# How To Raise The Dead

## (And Other Black Arts)

**by Mark Cooper**

About 4 years ago, I was a reasonably happy, skilled (in my opinion) COBOL programmer and system designer. The systems that are designed and written by the company I currently work for cater solely to a very specific market sector, and as such there is very little that the expertise within the company doesn't know about its target customers and their problems.

One day, an overpaid consultant bean-counter pointed out to the senior management and directors of the company that we weren't actually very profitable at what we do, and that the marketplace for the software we produced was changing and leaving us by the wayside. Our software looked old (pure text-based user interfaces) and was not attractive and sexy enough to catch the attention of a new breed of customers.

What could we do about it? The software was all written in COBOL, and despite the best efforts of many software tool companies to enable GUI capabilities in COBOL systems, we knew that we had other problems, which were far more deep-rooted. For example, every system we wrote was bespoke, but drew heavily on code written for previous projects (some of which were obviously more successful than others) – but we hadn't used the same code-base for all systems, so the enhancement and maintenance of those systems was always a very high-cost affair. In addition, systems were written without much in the way of design documentation – we didn't need it (so we thought) since all of the knowledge required to write a system was in the heads of the developers!

Systems were taking too long to write, and even longer to fix afterwards. We needed a new approach, one that would give us faster to develop, easier to maintain systems. The company embarked on a 2 year R&D programme to find the most suitable development tools and methods for a new generation of software.

During this time I saw 'the writing on the wall' for me as a 40 year old programmer. Whatever the R&D conclusions were going to be, if I remained as a developer I would have to 'go back to school'. The nightmares began – I would wake up sweating in the night from the aftermath of a dream where I'm in a lecture room with hundreds of kids with pony-tails and baseball caps calling me 'dad' and 'old man', laughing at me as I fail to understand the basics of whatever it is. I then made up my mind that the only way out for me was to become – a MANAGER.

So that's exactly what I did. I landed myself the role of Development Manager, and for a year or so became blissfully happy not having to worry about new technology and managed the lives and tasks of about 40 or so COBOL programmers.

Meanwhile, in a darkened room in another building, strange things were afoot. The R&D program had reached the conclusion that some language bearing a resemblance to C++, but with a name that was something to do with coffee had won the day and had been chosen as the path forward. I breathed a sigh of relief, secure in my management capacity that I wouldn't have to have anything at all to do with this thing, after all, if it were anything like C++ the code would be full of curly-brackets and be almost impossible to read and understand. We had got a couple of C/C++ programmers in the company - we kept them hidden away from 'normal' people as they even spoke differently. They used terms like Problem Domain, Class, Object and Implementation and spoke about such black arts as design patterns.

Life continued. Each day my loyal band of COBOL programmers would do some more coding (for some, unknown reason referred to as implementing by the warlocks in R&D). Then, the company decided it would like to actually write a system using this new thing (which I now knew the name of - Java), and embarked on a PR mission to extoll its praises to the rest of the company. Oh how I laughed! My COBOL programmers wouldn't be interested in this! They wouldn't know a curly-bracket if it smacked them in the teeth.

Mysterious things began happening. I got to work one morning to discover one of my best programmers missing. Someone or something had kidnapped him over the weekend. I raised the alarm, only to be told that the traitor had been lured by the powers of the Dark Side, learning Java in his spare time, and had volunteered for the new pilot project. I was distraught. I had a pep talk with the remaining programmers, and was even more surprised to find that several others had been tempted by these dark forces. Over the next few months I took to counting the empty desks every morning to find out how many programmers I had left.

One day my manager, the Technical Director of the company drew me to one side. 'Aha!' I thought, 'He's finally going to replace my missing programmers'. Unfortunately, this wasn't what he had in mind. He told me that the new project wasn't going so well, and seemed to lack direction.

'Get in there', he said. 'Find out what's going wrong, use your technical skills, which I'm sure you've kept up to date – use your experience!' I was crestfallen. How could I tell him that this was as alien to me as a pocket PC would be to the Dalai Llama!

Reluctantly, I moved out of my cosy private office, into a room full of trainee sorcerors. I tried to understand. I tried to learn the buzzwords, discovering for example that everyone hadn't suddenly had a windfall from a dead relative, just because they were all talking about inheritance. I even made half-hearted attempts to use an Object Oriented Design Tool. Eventually, with much kicking and screaming and gnashing of teeth we managed to get a system together. We even installed it with a customer, and it (for the most part) worked. I knew deep down that the system was written badly, and had been designed (if you could call it that) mainly by the programmers, who, having nearly all come from a COBOL background had no idea how to design an OO system.

The company directors saw what they wanted to see, and immediately began finalising plans for the take up of the new system for other projects. I had to do something, otherwise I knew we would be no better off than before. We would still be turning out systems based on a badly written code-base with little or no design documentation.

I casually mentioned to the technical director that it might be a good idea to get someone else to look at the system, someone from outside to offer an unbiased appraisal of what we had achieved. After some weeks of badgering, I finally convinced him to cross with silver the palm of a highly trained adept (contract an OO consultant) to cast his scrying bones over the results of our crafting (review the system).

The guy came in and spent a week looking through the code, the documentation and the practices we'd used to achieve the system. He and his scribes conjured forth the Booke of Counted Sorrows (he produced a bound, 30 page report). It wasn't **all** bad, we had after all managed to get a working system, and by all accounts most software companies embarking on their first OO system (or indeed any new methodology) end up throwing it away. The rest of the report was, as I'd suspected, **not** what the directors wanted to hear. The report basically recommended that we throw away much of what had been done, and start from the ground up using the correct OO design techniques.

'Do you understand all of this stuff?' asked my technical director, having leafed through the report and seen the implications. I had to admit that I didn't. I was immediately enrolled on to an OO design course. This lasted a week, and although it did take away some of the 'black art' mystique surrounding OO concepts, the course centred mainly on the use of a particular design tool and wasn't really what I needed. I was assigned the task, along with a couple of colleagues of designing a more generic system, one that could be modular and would be designed for re-use. For some weeks we laboured at the task, but without a sound understanding of Java (or any other OO programming language) it was slow going, and I wasn't confident that we were approaching it the right way.

About this time, a CV dropped on my desk. This usually happened whenever the HR manager was away for some reason. This particular CV had been read by her previously and discarded due mainly to the fact that the person it described wanted a ludicrously high salary (when compared with our 'usual' rates for developers). I read it several times, and even visited the guy's web page to find out more. The chap considered himself to be mainly a C++ programmer, but confessed to having 'dabbled' in Java, but his main attribute was years of experience in OO. I managed to convince the technical director that we needed to bring in some new blood. Someone experienced with the complexities of OO methodologies. He reluctantly agreed. Then I showed him the CV.

I peeled him off the ceiling a couple of days later, and suggested that we interviewed the candidate. I was relieved to find that the guy was still available, and arranged for him to be interviewed not by myself, but by the technical director and one of the senior developers who was by now our most experienced Java programmer (he'd been using it for about 6 months!). I was pleasantly surprised when it was announced that the applicant was successful. As I'd suspected, even though the guy only professed to have 'dabbled' in Java he still knew more about it than any of our people.

And so it came to pass that some weeks later this wizard arrived in my office. As befits most practitioners of his arts he seemed of indeterminate

age and came attired in his formal sorcerer's garb – black jeans, black T-shirt and long hair in a ponytail. When he spoke, he might as well have been speaking Greek. Still, he seemed to possess an air of knowingness and a good sense of humour – he even took it in good heart when my colleagues and I would mimic pulling dunces caps over our heads when he said something particularly unintelligible. He suggested several weighty spellbooks for me to read (some of which I did at least flick through).

It took a few weeks, but eventually he managed to grasp the necessary social skills to speak to us in language we could understand. It couldn't have been easy. He seemed as far up the evolutionary scale from us as we are from pond scum. We began designing the system again. The new guy was every bit as good as his CV, and we agreed we could see the way forward. I went on yet another OO design course and things were progressing nicely. We introduced sound design principles and reviews at each stage in the design process. We managed to borrow a couple of Java programmers from the original project and gradually began to introduce them to the new design techniques and our new architecture.

Then we got a hammer blow. The company board was becoming disillusioned by the whole project, and needed to see some tangible evidence that what we were doing was worthwhile. They had to make a commercial decision whether or not to continue with the project. We were given a customer prospect, one with a more limited set of requirements than usual, but for which we could write a system that would be in essence a cut-down model from the monster we were currently cooking up. The problem was that there was a ridiculously short timescale and the customer was particularly adamant about the delivery dates. As an added 'incentive', the customer's decision as to whether or not to give us a further squillion pounds worth of work rested solely on our ability to deliver this first phase, working and on time for October (by now it was June).

During the project meeting that followed, it became apparent that we were short of at least 2 developers. We managed to abduct one from another team (they did notice after some days, but we got away with it). That left us still one developer short. It was at about this time that someone really decided to rattle my coffin lid.

'Why don't **you** do some of the implementation', suggested our new guy. 'You really should know more about implementing Java systems anyway if you want to manage the projects'. I was horrified – **me**, learn Java! The mere thought of looking at all of those curly-brackets made me hyperventilate. To my utter dismay, he had even made the suggestion out loud, in earshot of other colleagues **and** the aforementioned technical director.

'What a good idea!' I said, noting the smile of approval on the tech director's face, and hiding my panic as well as I could. I was enrolled on a 'Java for non-C++ programmers' course the following week. The first couple of days passed pretty much in a blur, trying to learn new concepts like iterators and enumerators, and seeing beautifully constructed paragraphs of if statements contracted to a single half-line of gibberish. Things settled down after that, and despite my earlier misgivings I found that I was getting the 'feel' for it. The OO design concepts I'd seen before from the previous courses suddenly made sense.

I suppose that I was hoping that during my time away, the urgency of the project would have dictated that they take another programmer into the team and that I wouldn't actually be required to do any coding. Imagine my dismay then, when I arrived at work on the Monday following my course to find that I'd actually been assigned some classes to implement. 'You mean you want **me** to code those programs?' I said, clinging limpet-like to the terminology into which my addled old brain was still firmly anchored.

Fortunately for me, the package level and class level design that we'd done previously meant that all I'd really got to concentrate on was the code itself, and I even impressed myself by having almost completed my tasks (as far as I was concerned) pretty much ahead of schedule. As I've said before, someone rattled my coffin lid. Well, then it happened that this same person (our new OO wizard) tore my coffin lid off its hinges and applied the lightning rods to my neck-bolts.

'I see your classes are coming along nicely', he said, 'how are you getting on with their test-harness?'

'What?' I said, 'No-one mentioned a test-harness to me'.

'Then how do you know it will work then?' he said, smugly.

'Because I've written the code, and I know it will work!' I said, equally smugly. He gave me a look which said 'You really are pond-scum', and the URL of a website for JUnit (a set of classes for Java used for implementing test-harnesses), and politely suggested that I should read the information there and then talk to him again.

Even after reading the JUnit documentation and tutorial, I was still sceptical that it would apply to us. It was quite obvious that to be useful, the test-harnesses would be almost as difficult to write as the classes themselves. I expressed my concerns to the guru, explaining that we just didn't have the time to write all that extra code, we would have to rely on the final system testing. Then he pointed out that if we didn't have time to write test harnesses, we certainly didn't have the time to re-work any problems later, surely it was better to put the effort in while coding, and structure the test harnesses in such a way as to only test the critical parts of each package to ensure that future code additions and modifications didn't 'break' the build.

Reluctantly, I agreed with him, and proceeded to write a test harness for my new classes. As I'd surmised, this took me nearly as long as writing the code, but once I'd done it once and became familiar with the JUnit classes, further test harnesses became easier to write. The other developers were almost as difficult to convince as myself, but eventually everyone got into the idea that no task was complete without a JUnit test harness. Our guru put together an immensely useful piece of template code around which most of the tests could be written, and everyone was happy.

Everyone, that is, except the Project Manager and the directors. You see, also under the guidance of our new wizard, we hadn't even begun to write any user interface code. We had taken the approach that it was more important to get the server-side code written and stable before tackling the user interfaces. This was completely contrary to everything we were used to. Project milestones had traditionally been measured by the amount of visible functionality completed. Non-programmers only see this by the functions available from the user interface. One week before we were due to make the first delivery to the customer, nothing was visible.

I'm reminded here of a wonderful Dilbert cartoon, where he's standing in front of an audience of interested parties demonstrating the new prototype system they are developing. The screen he's pointing to is blank, but Dilbert gets away with it by saying "The User Interface hasn't been written yet, but if it had you would see a box here (points) and some text here (points) – and you'd be saying 'I've gotta have me some of that!'"

Despite this, our new OO guy was still confident. I tried to stay calm, and assured everyone that everything was going to plan, even though I was by now having serious doubts.

Thankfully, my doubts were unfounded, and the user interfaces required for the first software delivery were written in next to no time, due to the fact that all of the hard stuff was being done server side (where it should be) and worked virtually straight away. Thanks to the JUnit testing, all of the server code performed extremely well.

The rest of the project went equally well, and although we all had to put in some long hours, and we scraped a few of the delivery deadlines by a matter of hours. The customer was extremely happy, due to the excellent quality of the software (only a few minor problems arose during User Acceptance Testing). Management was extremely happy due to the timely release of resource on completion of the project (we needed very few of the original team to cope with the few bugs we did get). Most of all, the development team were all extremely happy due to (for once) being given sufficient design information to be able to do their jobs without constantly having to resolve issues.

I've learned more in the last 9 months than I had previously in 17 years as a COBOL programmer and 3 as a senior manager. More importantly, I feel like someone has 'given me my brain back', even though I never knew it was missing in the first place.

To any 40-something programmer reading this who still has misgivings about learning something new I can only say this – you may feel like you've become a Junior Programmer again for a while, but if you are like me (at heart **always** a programmer) you will get a **lot** of job satisfaction. I now get up most mornings, still with aches in my bones, but also with a spring in my step, because I know that at the end of each day I will be just that tiny bit wiser than the day before – a feeling I thought I'd lost forever years ago!

I still can't understand why OO developers felt the need to invent a whole new set of terminology for the subject when there was a perfectly good collection of gibberish already in use, but I suppose it makes sense to keep the mystique alive – it wouldn't do for those pesky **managers** to understand everything we do – would it?

*Mark Cooper*
*mark_cooper@bigfoot.com*



Reprinted from C Vu April 2002 / © Mark Cooper