BT DevCon 3

Paradigms of Programming Retrospective

This year at DevCon3, Anna Kennedy and I ran a session first devised by Steve Freeman and Michael Feathers when they presented the ‘Paradigms of Programming’ session at the BCS SPA2010 conference. The session is focused upon expanding knowledge and awareness of different paradigms which can be programmed in instead of focusing on the languages used- a point Robert Floyd made in his 1978 Turing award acceptance article when he wrote:

“...advancement of the art of the individual programmer requires that he expand his repertory of paradigms”.

To demonstrate this idea, 4 paradigms of programming were selected: Procedural, Object Oriented, Functional and Rule-based.

# The Exercise

The DevCon attendees were split into four teams to work together at a table. Each table was assigned one of the paradigms and the team used this paradigm to write a pseudo-code solution. They had 15 minutes to do this in before each team picked one person who understood their work to remain as the “maintainer”. Everyone else moved to the next table to begin working with the next paradigm.

The second iteration then began with the teams first being briefed by their maintainer on the thoughts and design of the previous team. The problem was extended to require more functionality and each team worked to add this to the program they had inherited, writing in the paradigm of the table they were now at.

At the end of this iteration a new maintainer was selected with the express rule that this could not be the person who had played the maintainer for this iteration, and the process started again.

As well as allowing a comparison between different paradigms implementations for the same problem, the exercise has some interesting features designed in by Michael Feathers and Steve Freeman.

The most notable is that after the 2nd iteration each team is maintaining code where the original developers aren’t available. This helps to demonstrate the intrinsic ease of comprehension each program had.

The tasks for iterations 3 and 4 also give great opportunities for reuse and can highlight the defects and merits of the design.

# The Results

It was interesting to watch how the teams progressed whilst performing the tasks and the effects that this had on code verbosity, maintainability and the time it took to complete the tasks.

It was especially interesting to see a couple of the participants who are not software developers joining in. These people seemed to have a much easier time switching between paradigms and remarked on how they enjoyed trying the different paradigms.

In contrast, some of the developers who would occasionally muse “How would we do this in <favourite language>?” seemed to find it somewhat more difficult. This is a perfect example of what happens if we make one language and the act of programming synonymous in our minds: we hamper our ability to think of other solutions outside of our ‘main’ language. I do not think this is a problem so long as we are mindful of how to implement other paradigms in that language, which is the main point I feel this exercise makes very well.

Outside of this macro-people watching, it was very interesting to observe the common themes which each team followed when working on a particular paradigm. Here is a brief summary of what we saw:

## Object-Oriented

The Object Oriented paradigm is certainly the one with which most of the attendees were familiar. This was reflected in the fact that whichever team was working on it, that table tended to complete the set task first.

The structure was easy to extend and reuse. It also made it easier to see the concepts of what the original designers had been thinking as they solved the initial requirements.

On the downside, the final code for the OO implementation was very long, partly down to the very-close-to-Java pseudo-code used and partly due to the relatively verbose code needed to add functionality in the later iterations.

## Procedural

The procedural paradigm was also familiar to our teams and as such it reflected many of the same benefits and realisations as OO. However, as the code grew phrases like “erm...this does some stuff” crept in more and more and pen-testing/debugging took more time as people struggled to follow the flow of execution and its resulting effects on the data structures being used.

Despite these drawbacks, the code didn’t get too big or become particularly unwieldy, though it was certainly the longest solution at the end.

## Rule-Based

This paradigm was relatively alien to most attendees. In general, it was agreed that this paradigm was pretty simple once you were able to get your head around how it worked. It was decided that this was probably the easiest paradigm to extend and it was very easy to implement this particular problem with this paradigm.

However, the discussion also raised the point that for a system or problem which required a larger number of rules, it might become unwieldy and almost impossible to debug as the composition of rules gets more complex.

## Functional

This was largely considered to be the hardest paradigm to work with – mostly due to people’s unfamiliarity with the paradigm. It was particularly interesting to note that when each team arrived at this paradigm, they tried to rewrite the existing code and start all over again as very few people understood what was going on.

The discussion around this paradigm raised the point that it is perhaps harder to decompose a problem into manageable chunks when using the functional paradigm, though no one was sure if this is an inherent issue with the paradigm or whether it is just a side-effect of most participants being used to decomposing the problem into objects and items rather than events or functions. The difficulty I had in finding a clear and concise definition of functional programming online also reinforces our results, though I am unconvinced that it isn’t down to the same issues of the prevailing mindset decomposing problems to items/objects.

Another issue that this paradigm specifically faced during our session was that the syntax was very hard to understand.

# Conclusions

So what did this exercise demonstrate?

The primary lesson to learn here is the reminder that we can solve most problems in a variety of different paradigms. Each paradigm is appropriate for different scenarios and, whilst some languages are certainly written with one paradigm in mind, we can write code in a style approximating different paradigms in any language. As Floyd suggested, we become better programmers by expanding our repertoires of paradigms.

We also demonstrated that it can be difficult to maintain code written by others in styles with which we are unfamiliar. Whilst this could easily be used as an argument to support the idea that we should not stray from the core paradigm of the language we are writing in, I believe that we should hold each other to a higher standard and expect a good programmer to be at least slightly familiar with these alternate paradigms. If we avoid using them because someone else may not understand them we can fool ourselves into missing better solutions and enter a vicious circle of neglecting the less used paradigms.

Hopefully it serves as a reminder that, whilst learning new languages, API’s and design patterns are important for us to be more productive, we may (and many of us have) narrow our fields of vision to only look at procedural and OO paradigms. There is a lot of value and many possibilities in other paradigms which do not get very much attention outside of academia.

# Thoughts for future workshops

There are a few things I would like to try or would suggest trying in the future if this session is run again:

## Changes to the running

* One defining issue with our session was that, whilst Steve and Michael’s original session had 3 hours at SPA2010, we attempted to fit it into a 1.5 hour slot. This meant that the summary was compressed but otherwise it worked well. I would suggest in future at least 2 hours was given (10 minute intro, 4x20 minute iterations, 20+ minutes for discussion and summary)
* Bolstering the documentation on the paradigms with more examples, especially the functional handout, to alleviate some of the trouble with understanding the paradigms.
* We had some issues with the 4 teams having ~15 members per team. Due to a lack of space we could not split it into 8 teams of ~7 each, but I think this size would have been far better to make sure everyone could get involved.

## Changes to the exercise

* Try a 5th iteration where each item in the list needs to be manipulated. This will negate some of the advantages towards the non-functional paradigms and give each team a chance to see what their original foundation work became.
* Change the rule-based paradigm for constraint-based and see how easily the problem is solved.

The feedback from the session was very positive and we hope all those taking part enjoyed it!

If you have any feedback you feel we’ve missed, or any questions, please contact either Adam Whittingham or Anna Kennedy.