CECUBE NEWSLETTER – Software Components

Component re-use has been a hot topic but expectations are not being fulfilled

One reasons is that although the IT industry is evolving very fast, it takes time for new methodologies and technologies to mature and stabilise. After all these ideas are relatively new compared to many other industries. Trains have been around for a hundred and fifty years but how many really big technological advances have there been during that time? Compared to the software industry the pace of change is relatively slow. There is also huge amount of legacy code in almost every large organisation that does not fit easily with new architectures. It will take time before the majority of the programs are written in a way that promotes re-usability.

Another problem is that it is not easy to find a suitable component. Components need to be built and documented in a way that makes it easy to establish what it can do. The documentation within a small component should be so clear that you can see in minutes what it can do and if it of use. If the component is to be sold, it requires even clearer formal documentation, including information on the operating systems and development environments with which it has been tested. Succinct functional description is needed when searching for a component and detailed technical information when you have found it.

Not a new idea

Code has been re-used from the dawn of computing. Using code snippets has always been a good and widely used way of learning the tricks of programming. Any accomplished practitioner has learnt and developed skills quickly in this environment, supplementing or even replacing book and course work. Many skilful programmers have shared their knowledge with the whole community. Open source projects have been a popular way of providing code for re-use. The popularity of Linux would be much less, and may not have existed at all, if it did not give away code for re-use.

> *"Re-usable code should not be thought of as a 'car-boot' purchase, but a better product at a cost-effective price."*

Re-usable components are a variation on the traditional code re-use concept. The component still comprises lines of code but it usually has been compiled, tested, and is ready to use. Some components include the source code for making changes or seeing how it has been built. Re-usable components have a defined interface that allows other programs to use it and that makes it possible to use the component from other programs even if they are not written using the same programming language. Compiling source code on a different environment and compiler from which it was developed originally rarely works without modification. It still requires more rigorous unit testing on your system than compiled components that have been tested already by their creators.

Develop from scratch or use existing components?

Before a line of code is written, check for existing components; ideally this should be performed much earlier during the designing phase. In most cases existing components cost much less than ones developed from scratch. What kind of component could you build for a few hundred pounds?

> *"Developing components designed for re-use still takes more time and effort but the benefits are long lasting."*

Off the shelf components have often been tested in real life use and besides reducing development efforts they also reduce testing efforts. Bugs, compatibility problems, development needs are easier to find when code has been widely used. It is estimated that more than half of all programming efforts could be saved with careful study and use of existing components, providing an opportunity to improve productivity. This requires changes in attitudes at an organisational and technological level. Re-usable code should not be thought of as a 'car-boot' purchase, but a better product at a cost-effective price.

In some cases there will be no suitable components or there are other reasons to start from scratch. In these cases it should be common practice to think whether there might be a need for this kind of component in other development projects in your organisation or perhaps outside of it. Many organisations regard software with low IPR value as sacrosanct, so missing a technology transfer market opportunity. Developing components designed for re-use still takes more time and effort but the benefits are long lasting. Some companies save a lot of time and money, while others make nice profits by selling a block of code again and again. Once the component has been built it is cheap to manufacture and distribute. Using suitable application architecture and designing components with re-usability in mind will pay off.

Driving Technology

Developing re-usable components has always been more difficult than writing traditional code but this has changed a lot recently. Object oriented modelling and programming (OOP) encourages the writing of more robust code that is easier to re-use. A component implemented object oriented program is automatically a good candidate for re-use. In the real-time programming environment OOP is not easy to adopt into an environment with strict execution time constraints. However, the concept and practise of re-usability is still prevalent. Well-designed code can legitimately add a reference to a component and start using the provided application services.

> *"Success is based on driving the available technology – not being driven by it"*

Development tools, application architecture, frameworks and platforms have evolved to improve support for the component model. The NET framework is probably one of the biggest and most well known steps towards to more common re-use of existing components. This is not just because it is technically big step forward, but because it provides a platform and tools for millions of developers familiar with Visual Basic, C/C++ and other development tools or efficiently generate mixed language programs. Java 2 Enterprise Edition (J2EE) is another popular development architecture providing strong grounds for re-usability and integration with web based technologies. Component repositories and libraries are being established; modelling and design tools make designing and documenting components easier; the future looks optimistic from technological point of view. However, the player who benefits will prove success is based on driving the available technology – not being driven by it.

The NET framework has many new features and it takes so much effort to learn it that developers are not able to focus on anything else. This is one reason why the component market has not been more active. Since 1990 I have been actively creating railway application code in a component based way for re-use. A component can have as little as 20 or more than 500 lines of code. Code flexibility and adaptability is the key to re-use in either a bespoke or OOP environment. Generally speaking it is amazingly easy to use existing components. What could be a better way of regaining development time than smart re-use.

The Way Forward

Time will take care of some of the problems. The framework will improve further and new code will eventually replace the old. Attitudes and understanding will improve with time so that benefits are too significant to be ignored. More and more good quality components will be published, and as the component business grows and competition tightens, it will force vendors to provide easier ways of finding suitable components.

Developers and managers need to take time to learn what kind of advantages may be offered by component re-use. Changing practices to use them will in most cases help in developing better programs, improve portability and extend productivity and profitability.

Cecube Software is currently providing a wide range of traction and railway modelling components. Some of them are relatively old but usable with legacy or new applications. Others are new and built with recent technologies. There are both general and specialised components to fit the specialist railway application. For a broader overview and examples the reader is referred to the alternative discussion with examples found at http://www.cecube.co.uk/software/software.htm.