Data-driven Programming

By

Johan Nel

A series of articles explaining the principles

Article 1: Concepts and theory

November 2014

Table of contents

1. Fruit for thought

It appeared many times to me that software development is chasing it's own tail. Just as I had my problems solved in Clipper, users demanded that my applications be Windows orientated. So I have spent many hours rewriting and on the way enhancing my front-ends to make users happy in my language of choice Visual Objects. Did my newly developed application do anything more than the old MS-Dos Clipper application? Unfortunately my answer is no. The business logic stayed the same, but users were happy that they can now Point and Click with a mouse instead of pressing some hot-keys to activate actions. Well a new kid arrived on the block DOTNET and again my software users demanded that my applications are outdated and I need to re-engineer. Needless to say, I had a new toy called Vulcan.NET and I persue to make my clients happy. So what new functionality did all these changes bring about? Firstly and I will generalise, my software users still want to be able to Insert/Update/Delete their data in a [relational]database. Secondly they still have an ever growing demand for reports. Thirdly they have an ever growing demand for some business processes that need to be executed that cannot be categorised into any one of the above demands. Altough a lot of readers might get the impression that I am negative, on the contrary I am quite positive, most of these changes brought challenges to learn new technologies, however from an end-user perspective the way we do accounting (business data management) have not changed in a thousand years, except that we now do it electronically. So my statement is rather, what can a DOTNET based accounting software application do that cannot be achieved by a Clipper developed one? I would like to rephrase the above rather as follows: "If the time spend on re-engineering due to platform changes, were spend on enhancements to end-user business demands, how much more could we have achieved?"

This series of articles will look at a way of shortening the cost and time of re-engineering efforts due to technology advances using a data-driven concept. I leave you with the quote by Appleton (1983) before we delve into the concept and theory:

> *"The nature of end-user software development and maintenance will change radically over the next five years simply because 500 000 programmers will not be able to rewrite $400 billion of existing software (which is hostage to a seven- to 10-year life cycle). They'll be further burdened by those new applications in the known backlog, as well as by those applications in the*

*hidden backlog. To solve the problem, dp (*data processing*) shops must improve productivity in generating end-user software and provide end-users with the means of generating their own software without creating anarchy... The answer to this is a data-driven prototyping approach, and companies that do not move smoothly in this direction will either drown in their own information pollution or loose millions on systems that are late, cost too much, and atrophy too quickly."*

2. Data-driven programming

Information system development primarily aims at managing enterprise data and deriving information from it. The *information age* has driven industries to better exploit the growing amount of, and need for universal access to critical corporate information and data.

Traditionally, end-user application software systems were and even today are developed by means of a functional decomposition strategy. This approach is still used in the in-house system development environment. Requirements are expressed in terms of system outputs and are predetermined. A software application is then constructed that will best satisfy those predetermined requirements. Few information system projects however, come in on schedule or within budget, and those that do are often characterised by extremely high maintenance costs (Appleton, 1983).

This is echoed by Kalka (1995), in an article called "*Sustaining the re-engineering business: the open information technology initiative*". Even today problems are still encountered in systems development:

- Research shows that despite an overall satisfaction with current re-engineering efforts, the benefits actually achieved are not meeting original expectations.
- An estimated 70% of first-time Business Process Reengineering (BPR) efforts fail.
- IBM (Anon., 1994) has found that less than 10% of the Fortune 500 companies are using client/server computing for mission-critical business functions, and more than 80% of existing client/server projects are experiencing cost overruns and schedule slippages.
- In the same study by IBM, it was also found that only 20% of time is spent on development, while the rest are spent on maintenance of systems.

The last finding is alarming, considering that software tools exist to presumably improve overall productivity in systems development e.g. Business Process (BP) Modelling, Computer

Aided Software Engineering (CASE), Fourth Generation Languages (4GL), Object Orientated Programming (OOP), Rapid Application Development (RAD) Tools and Relational Database Management Systems (RDBMS) with its client/server (CS) and distributed computing environment (DCE).

The fundamental aspect of functional decomposition lies in the assumption that requirements can be precisely determined before system construction is attempted. In contrast, precise requirements are not always definable before system construction. Requirements can develop in parallel with coding rather than serially. The drive behind all systems development efforts are still driven by the Information Age i.e. to exploit the growing amount of, and need for universal access to critical corporate information and data (Appleton, 1983). It is advocated by Appleton (1983) that data-driven applications, also known as active meta-data repositories (Appleton, 1999), are the only method that can solve the problems inherently found in the use of functional decomposition models to determine the need of end-users.

2.1 Scenario

In order to understand the concept of data-driven applications, a big picture approach to application development and maintenance should be assumed (Squires, 1992):

- A network supporting multiple applications over multiple file servers.
- A backlog of many applications that still need to be developed.
- Ever increasing demands by users of these multiple applications for new functionality to be added to the existing applications.
- A shortage of skilled system developers and programmers.
- A large turnover of staff.
- A small budget.

2.2 Managing applications

It is of no surprise that users are normally not satified with their applications. Not only are some needed functionalities not provided in applications, but most of the time, added functionality opens the horizon for a broader scope of possible things to include. In many cases, new functionality can have major impacts on a system. Provided that the functionality follows the current system methodology, it is relatively easy to implement and make the necessary changes for it to be incorporated, but if the new functionality cannot be

implemented in the current system methodology, cost overruns and schedule slippages can be experienced (Appleton, 1983).

The first test of a data-driven application is whether users can "have their own way". This does not imply that users are left on their own to develop new, or change and modify existing applications, and by doing so, complicating standardisation and creating anarchy or becoming programmers. It only implies that users can through guidance, modify applications to suit their specific needs, helping system developers with their burden without the need for different site specific versions of a system (Squires, 1992).

The management of system development and maintenance are inherent in any IS department. Not only are they involved in providing the necessary infrastructure for efficient system operation, but also for the management of the data and information flow in the company. IS management should also have many questions regarding the relationships between different systems and users interacting through those systems with the data and information. The following are examples of the type of questions to be answered by IS management (Squires, 1992):

- Which departments currently have access to a certain table?
- What indices belong to a certain file?
- How many times is a certain index being used?
- Who are the users having access to a certain application?
- In which tables does a certain field occur?
- How frequently is a certain view used?

In order for the IS manager to ask all these questions and have readily available answers to them, these system definitions should be stored in an appropriate environment and be maintained during system development. Managing information systems is not only about asking questions regarding these systems, but is also about changing these systems. When it is found that only one user that has resigned uses a certain index, managers should be able to delete it. Normally it would involve an application to be run, that specifically address this issue. If the details of all applications are adequately externalised and placed into a single, relational data dictionary, it would be possible to use this environment to manage and maintain all business systems information.

Using this methodology, it is possible to define and develop a data-driven environment. It offers system managers the same tools generally offered to end-users, the only difference being the data. Where users browse financial data, systems managers browse dictionaries.

Data-driven applications can be considered an elegant way to develop and manage multiple applications (Squires, 1992). It assumes that no application is an island in a company and that large segments are shared across applications. Due to system details been stored in a data dictionary, it is possible to use this dictionary during application execution.

The following definitions can be made (Squires, 1992):

- Applications are merely records in tables in the data-driven model.
- The data-driven technology is a methodology for stripping every detail of an application from the .EXE and the placement of these details in relational tables external to the application. In normal programming practices, an application's detail is within itself. It is all hard-coded. The data-driven environment, is different. None of the detail of an application is known at development time. The run-time environment will only collect the details at run-time.
- It provides maximum flexibility and provides the ability to tailor applications on the fly.
- This is in contradiction to the functional decomposition methodology normally followed in systems development. Level by level it is possible to reduce the whole of an application to mere records in a relational dictionary. It implies that an application is nothing more than records and fields in a database. These records and fields can be read into abstract data types called meta-data (Appleton, 1983) during run-time.
- The programmer's task is enhanced to create generic code. Coding should be done on an abstract level. At this level no detail resides. It only has the necessary functionality to get the details at run-time in the relational data dictionary.

The following assertions have been made (Squires, 1992):

- Applications are primarily data.
- Data best resides outside an application.
- Functionality can be reduced to records in tables.
- Loading an application is merely reading tables during run-time into meta-data application variables.
- Most changes to an application's behaviour can be made externally without recompiling the application.

The data-driven model manages applications as tables, records and fields, and not as source code. The information system manager takes ownership of the corporate data dictionary’s database maintenance. Application management is merely database management.

Once application-level details are externalised into the data dictionary, it is possible to develop a "super application" which manages and also performs the tasks of all applications.

3. Literature cited

Anon., 1995. Observations. IBM presentation at IBM University in Philadelphia, PA.

Appleton, D, 1983. Data-Driven Prototyping. *Datamation*, Vol. 29(11): 259-268.

Kalka, RA, 1995. Sustaining the re-engineering business. IBM Solution Developer Support: http://www.developer.ibm.com/library/ref/sustain_reeng/index.html.

Squires, L, 1992. Data-driven programming. IN: Proceeding of “The 1992 Clipper Devcon”. 23-25 March 1992, Johannesburg, South Africa.