

# Migrating to Web Services A Research Framework

by Harry M. Sneed  
ANECON GmbH, Vienna  
harry.sneed@anecon.at

**Abstract:** In this short discourse *the forces driving continuous software migration in industry are identified and explained. It is pointed out that users have always been obliged to migrate their systems for both technical and business reasons. The latest migration wave to web services is being driven by both technical and business forces. Although it is particularly drastic, this latest migration can be seen as yet another phase in the continuous evolution of corporate IT systems, an evolution which shows no sign of slowing down. Five alternates to mastering this particular migration are defined and discussed. The paper ends with a proposed framework for research on the subject of migrating to web services.*

## 1. Forces driving Migration

In principle, IT-users are in a constant state of migration. Hardly have they succeeded in moving to a new technical environment before that environment has become obsolete and they are faced with the necessity of moving again. There are two driving forces in this migration movement.

- one is the constantly changing IT technology,
- the other one is the constantly changing business world.

There is, of course, an intricate relationship between the two issues making it difficult to deal with them separately, but this separation of concerns is essential to understanding their interdependence. [1]

### 1.1 Technology Change

The IT technology is changing at a rate of once every five years. Earlier in the 1970's and 1980's the rate of change was every ten years. The reason for this shortening of the technology life cycle is not only to provide the users with more functionality, but also to satisfy the capital market. Software technology vendors are under extreme pressure from their stockholders to increase their sales and to raise their corporate profits. Until now, the only way to do that was by introducing new products and convincing users to buy them. [2]

The software vendors, supported by the technology freaks from the academic world have cultivated a

mania that anything new is beautiful and that everything old is ugly. To be regarded by his peers or by other IT users one has to be on the leading edge of technology. Those, who remain true to an older technology are treated with disdain and even ridiculed. This applies to individuals as well as to organizations. Both are being manipulated by vendor marketing and the need to keep up with their neighbors. Whether the new technology really improves their performance is open to question. Nicholas Carr and other experts from the business world have their doubts. [3]

The fact is that IT- users feel compelled to move to a new technology even without evidence that it will lower their costs or increase their productivity. Like limericks they are herded together and pushed into an unknown future, blindly confident that they are on the right path. No self respecting IT user would even consider straying away from the herd for fear of being isolated and considered exotic. They seem to accept the fact, that migration is inevitable.

With increasing sophistication of new technology, the benefits of using it have become increasingly blurred. The advantages of the new technology over the previous one have become less and less obvious, so that the vendors have to spend more and more money on marketing to convince the users that the change is good for them. The notion of return of investment – ROI – has become a key issue in introducing new technologies, but more often the calculation is manipulated in favor of the new technology. [4]

In the early days of computing migration was driven by hardware change. The software and the programming languages were more or less dependent on the hardware they were running on. The costs of the software were low compared to the other costs. Fortran and COBOL programs could run in both, batch and dialog modus and could process punch cards as well as magnetic tapes and removable disc units. In recent times technology change has become more of a software issue. The programming languages and the middleware are now the driving forces for migration.

This shift of emphasis began in the 1990's with the move to client/server systems. Here, the change was

both, hardware and software. Every user department wanted to have its own local computer with its own terminal work stations, but still the local computer should be connected to a central host computer somewhere, with a global database. Connecting the many nodes in a client/server environment became a major problem and led to a market for middleware. In this distributed hardware environment it became necessary to also distribute the software and this made it necessary to shift the programming paradigm from procedural to object-oriented [5].

The primary costs of the client/server migration were software costs. The existing mainframe programs had to be converted or rewritten. Many had to be downsized to even fit into the storage of local servers. [6]

The migration from the mainframe to client/server systems entailed many complicated techniques such as scrapping of user interfaces, hierarchical to relational database conversion, program downsizing and procedural to object-oriented language transformation [7]. Many compromises had to be made in order to push the migration through in an acceptable time with acceptable costs. Often good software had to be perverted in order to satisfy the requirements of the new distributed hardware environment. Much effort was spent on objectifying the systems, i.e. transforming procedural programs into object-oriented components. This was an extremely difficult task, which was never really satisfactory solved despite the fact that a lot of research went into it. [8]

Most users were content in the end to re-implement their user interfaces and to convert their data to relational databases. Very few users ever succeeded in redeveloping their old applications in a truly object-oriented way. This is because they were never able to recapture how the legacy systems operated in detail. The costs of re-documenting a legacy system down to the level of elementary operations and then re-implementing in a different architecture were prohibitive. Thus, in moving to the client/server technology, the quality of the existing software in the business applications actually degenerated as a result of architectural mismatch. [9]

No sooner had the IT-user community completed the migration of its software to a client/server architecture, then the next wave came along in form of the internet. This migration negated many of the aspects of the client/server migration. Now it was important to have all of the components on one common machine, so that they could be readily accessed by all the internet users. This often meant moving the distributed software back to a single host computer. The object-oriented nature of programs remained, but the methods now had to be

directly accessible from outside. This destroyed the purpose of many a class hierarchy. Besides that, the client user interfaces, implemented with some GUI tool kit had to be replaced by web pages in HTML. What is more, the distributed data had to be collected again in a common generally accessible database. [10]

Most of the IT-users had yet to complete the migration from client/server to the internet when the latest technology, namely service-oriented architecture came along. Now it became necessary to break the software up into service components stored on network servers with a standardized web service interface. The client software should be replaced by business process procedures programmed in some kind of Business Process Execution Language – BPEL - which steers the process, displays and accepts data via cascaded style sheets and invokes the web services. Now every local process has to maintain its own state as well as its own data, since the web services should be stateless and without access to their users own database. Service oriented Architecture requires a radical break with past technologies and a complete revamp of the enterprise applications. [11]

SOA is now the third major technology paradigm change in the last 15 years, changes driven by the software industry's insatiable need for new revenues. The IT users are actually victims of a volatile IT industry, intent on using them to sustain revenues by constantly introducing new technologies. If a user is to keep up with the ever changing technology, then he must be in a constant state of migration. In fact, technical migration already has a permanent status in most large corporations, having established organizational units just for that purpose, not to mention the many software houses, including those new ones in India, which thrive on such technical migrations. [12]

## 1.2. Business Change

Besides the pressure for technology change, there are also those forces driving business change, Organizations have gone from the classical deep hierarchical structures of the 1970's to the distributed business units of the 1990's to the now lean, process-oriented and fully computer-dependent, networked organizations of today. Of course, the changes to the business side are tightly coupled to the changes in IT-technology. Without the internet, it would not be possible to distribute business processes across the globe, just as it was not possible to break up businesses into individual business units without having them connected via common database servers in a client/server hierarchy. In this respect, technology and business changes complement one another. [13]

Business change is driven by the desire to make organizations more cost effective. The business reengineering revolution propagated by Hammers and Champy had the goal of structuring the business according to the business processes. [14] Rather than splitting up the responsibility for a particular process, such as order processing among several different units, each with a different specialty, one unit was to be responsible for the whole process from beginning to end. This is best illustrated by the handling of passengers for an air flight. Earlier, there was one employee outside to check you in. Another employee was inside to put you on the plane. Today, there is one employee who checks you in and who accompanies you to the plane. This employee is responsible for the whole boarding process. The parallel process of handling the baggage is the responsibility of others.

By fixing responsibility for a business process to one unit or individual, it is easier to ensure accountability and to measure performance. It is obvious that the business unit responsible for a local process does not want to be dependent upon a global IT-department to accomplish their job. The responsible manager wants to have his own local IT which he can control. The distributed systems architecture offered this solution. In the end each business unit had its own local server and its own team of IT specialists which could be used on demand. The productivity of the individual business units went up but also did the total costs of ownership. The costs of maintaining different solutions for each separate activity became in the end greater than the costs of sharing common resources. [15]

The concept of a service oriented IT promises to remedy this problem, It is an attempt to unite the business concept of distributed, independent business units, each managing a different process, with the concept of using common IT-resources, which can be applied to many processes. Service Oriented Architecture is in fact more a business issue than it is a technical one. Today's business world requires extreme flexibility. The business processes must be continually adapted to satisfy changing customer requirements. There is no longer time to initiate extensive and costly development projects. Even if they are agile, they still require time until they deliver a workable product and the outcome is still uncertain. Useful software requires time to ripen, time which most struggling business units no longer have. There is a pressing need for software on demand. This means, that the basic functionality should be available before it is even required. The business users need only select from a large catalog of software services – the UDDI – those functions which they require for their latest business process and to prepare the control procedures for invoking and linking the ready made services, using a business process language. [16]

## 2. The Emergence of Web-Services

This changes the role of the company IT from a software producer to a software broker. It is the task of the IT to see that the required services, i.e. the software components behind the service, are available when they are needed. This means, setting up a warehouse of reusable software components, a perennial vision of the software world which began already with the FORTRAN common subroutine libraries in the 1960's, progressed to the PL/I build-in functions and the macro libraries of the 1970's, continued with the reusable business modules of the 1980's and cumulated in the class libraries of the 1990's.

It has always been a goal of the software industry to reuse as much as possible of its proven components. This has less to do with the costs of development than with the high costs of testing. It requires an enormous effort to detect and remove all of the faults which a piece of software might have and to demonstrate that the software does what it is supposed to. For that the software has to go through a long ripening process. It would be irrational not to want to reuse that which is already proven. In this respect, web services are just another variant of the reoccurring reuse theme. [17]

What is different about web services is that they can be physically located anywhere in the World Wide Web, that they have a standard accessing interface and that their implementation is independent of the environment they are operating in. Earlier reusable components such as the classes in a common class library had to be compatible with the classes using or inheriting them. The same applied to standard business modules. With web services, this is no longer the case. They can be implemented in any language, even in basic Assembler language, and in any operating system as long as they are able to service their interface.

The use of a message oriented interface with XML and SOAP makes the implementation of the service software independent of the clients. The Web Service Definition Language –WSDL – is an XML-based language for interpreting the messages which are passed between the service and the clients. The service software must only be able to read and write the WSDL messages. That the web services can be located in any node of the network is made possible by the HTTP addressing mechanisms. Each service has its own unique address and can receive messages from any other node in the network which has access to that address. [18]

Other than these internet specific features, there is no real difference between web services and the standard subroutines of the 1960's. They process a

set of arguments they receive to produce one or more results which they return. If they are stateless, they will have no own memory, meaning whatever intermediate data they have collected will be forgotten when they are finished. That puts the burden of maintaining the correct processing state on the client. If they retain the data state, they become much more complicated, because in that case, they have to distinguish between their clients and keep their data separated. This too is a reoccurring problem. It existed as well for the teleprocessing monitors of the 1980's such as CICS and IMS-DC and has been solved in many different ways. [19]

The problem with web services is the same as it has always been with reusable standard modules and that is the question of granularity. How much functionality should one web service provide. For example, the web service could be for maintaining a bank account with all of the functions that go along with that: Opening the account, making deposits, making withdrawals, transferring funds, computing interest, creating balance notices, checking credit limits and closing the account. This will, of course, require a highly complex interface. On the other hand, the web service could be restricted to simply computing interest. That would lead to a very simple interface. [20]

The business users would prefer the latter solution – the smallest granularity – because it gives them the maximum flexibility. The IT department would prefer to offer the first solution, because it is easier to manage. Having many messages also puts a big load on the network since the many messages have to be passed back and forth between the web clients and web servers. This was not the case with standard subroutines or base classes running in the same address space as their users. That leaves two problems to solve

- The problem of maintaining state
- The problem of determining granularity.

The research community has to deal with these two issues and come up with viable solutions for web services to be adapted in industry. Not only must state be maintained, but it must also be secured. The proper level of granularity is context dependent and must be determined for every particular situation. There is no global solution. It will not be easy to find proper solutions to these problems.

A third problem is how to come up with the web services in the first place. Unlike babies they will not be brought by the stork and deposited at the front door. They have to either be purchased, rented, borrowed, recovered or built. These are the five basic alternatives to providing web services to be discussed here.

### 3. Providing Web Services

The five sources of web services are, as pointed above

- To be purchased
- To be rented
- To be borrowed
- To be built
- To be recovered

#### 3.1. Purchasing Web Services

Web services can be bought from a software vendor. Most of the classical ERP software package dealers also offer their components as web services, so any IT user can buy them. This applies not only to commercial business software but also to engineering and scientific software, which can be purchased off the shelf. The advantages of such ready made off the shelf web services are:

- they are readily available,
- they are well tested and relatively reliable,
- they are supported by the vendor [21]

The second and third advantages should not be underestimated. It requires a significant amount of effort to test even a simple web service in all the variations of its usage. It is also comforting to know that the web service will be maintained on a regular basis, that it will be updated and that the customer does not have to deal with these issues.

The disadvantages of off the shelf web services are:

- they are usually very expensive,
- they are restricted in their functionality,
- the user has no possibility of altering them,
- they are most often too big and monolithic

The biggest disadvantage is the lack of flexibility. Using large, monolithic web services such as a ready made billing system or a credit checking package is like building with prefabricated walls. The IT user has to build his business processes around the purchased web services. As such, the web services determine how his business processes are structured. The user is not only purchasing software but the whole business process as well.

For some IT-users this may be a blessing. They don't have to spend time and effort in designing business processes, but they also lose the main advantage of web services and that is flexibility. They might as well buy the whole business process, as they have done in the past. There is no sense in moving to a service-oriented architecture. For other IT users keen on customizing the business processes this is an intolerable restriction. What will become of competition if every competitor is using the same business process?

### 3.2. Renting Web Services

An alternative to buying web services is to rent them. Many ready made software vendors such as SAP and Oracle are now working out plans to make their services available on a rental basis. Rather than having to purchase the software packages and install them in his environment, the IT user has the option of using only those functions he requires, when he requires them, i.e., on demand. He then pays only for actual usage. This business scheme has many advantages over the purchasing scheme:

- One, the user is not obliged to install and update the web service on his site
- Two, the user is always working with the latest updated version
- Three, the user only pays for what he really uses

Owning software is not always advantageous to the software user. He has to deal with the total costs of ownership. Not only does he have to install and test the software in his environment, but he also has to install and test the new releases. Maintaining the software in his environment has a high price. The advantage is that he can customize the software to satisfy his particular requirements. The user is not obliged to adjust his business process to fit the standard software, which is the case when he is only renting.

The same applies to web services. If he buys them, he can adapt them. If, however, they are rented, he must use them as they are. If the granularity of the services is small enough, this will not be too great a problem for him. He can build them into his business processes like gravel stones in a cement wall, but if they are built like slabs of ready made concrete, then he has to adapt his construction plans to fit them in. On the other hand, the user is free from having to constantly install and test new releases. [22]

### 3.3. Borrowing Web Services

Taking web services from the open source community is equivalent to borrowing them. Proponents of open source are those who would prefer to let others do their work for them and then to take it over with or without changes. On the one hand they don't want to pay for it and on the other hand they don't want to develop it. Open source web services are seen as public property which anyone can use.

There are two issues here – one moral and the other legal. The moral issue is that software, including web services, is intellectual property. Someone has sacrificed his or her valuable time to construct a solution to a pressing problem. If this solution

happens to be valuable to someone else than that person should be willing to pay for it. Otherwise he is violating the principles of a free market economy. So in this respect, the use of open source web services is questionable and not in tune with the society in which we live.

The legal issue is the question of liability. Who is liable for what the borrowed service performs? Of course, it can not be the authors, since they are oblivious to where and how their intellectual property is being used. Therefore, it can only be the user of the borrowed good. In taking web services from the open source community, the user is free to adapt the source to his own particular needs, but then he also must assume responsibility for its correctness and its quality, meaning to test it thoroughly for all possible usages. Most persons do not realize that testing software is equivalent in time and cost to developing it. And, if the source is unfamiliar to the persons who must adapt and correct it, it may be even more expensive than if they were able to write the software themselves. This way they would at least be familiar with the code. Numerous studies in the software maintenance community have proven that the greatest effect in maintaining software is the effect spent on trying to comprehend it. [23]

Code comprehension is the biggest barrier to using open source code. So, users should think twice before they start borrowing web services from the open source community. This could turn out to be a Trojan horse.

### 3.4 Building Web Services

Web Services can, like other software packages, be developed by the user organization itself or by a contractor working for him. The difference to conventional software applications is that web services, if they are defined correctly, should be much smaller and easier to build. The other difference is that web services are common property of the enterprise that provides them, i.e. they belong to all of the departments of that enterprise. That is a serious break with the past tradition of how software in an enterprise is financed.

In the past the IT-Department has been considered to be an internal software house with the mission of providing services to the various user departments. If the marketing department wants a new customer relationship system, it commissions the IT-Department to develop or buy one for them. If the logistics department wants a new order entry system, it commissions the IT-Department to build it for them. It is the user departments that have the power over the purse and who are only prepared to pay for something that has a direct benefit to them.

In the case of web services, it is not clear to whom they belong. Anyone within the organizational network can access them. So who should pay for them? The fact is that it takes a lot of effort to plan, develop and to test good web services. They should be more stable and reliable than the restricted use systems of the past and they should also be reusable for different purposes within different contexts. Like other user systems web services will cost three times more than ordinary single user systems. User departments are very reluctant to fund projects which are not dedicated to their requirements and which promise only a long term benefit. If they have a problem to be solved, they want it to be solved immediately and directly, i.e. it should be customized to fulfill their particular requirements and nothing else.

Developing web services is a long term investment. [24] It will take at least two years before enough services of sufficient quality can be made available to the user business processes. It is questionable if the users are willing to wait so long. The benefits of using self-developed web services will only become visible after some years. In the meantime, the IT-Department must sustain the continuity of the existing systems. This puts a double burden on the organization. For this and other reasons, developing one's own web services may not be an attractive alternative. The biggest barrier is the time required. The second is the question of financing.

### 3.5 Recovering Web Services

The fifth and final alternative is to recover web services from the existing software applications. It is true that the existing software may be old, out of date and difficult to manage, but it works. Not only does it work, but it is also adapted to the local organization. It fits to the data and the environment of the organization. So why not reuse it. The goal should be not to use the existing applications as a whole, since they would probably not fit into the new business processes, but to extract certain parts from them. These parts could be methods, procedures, modules or components. The important thing is that they are independently executable. For that they have to be wrapped. Wrapping technology is the key to reusing existing software. It is not so important what language the existing software is written in, so long as it is executable in the server environment. [25]

Since requests to web services can be redirected it is perfectly legitimate to have different hosting servers for the different language types. Thus, one could have COBOL and PL/I services on one server and C/C++ services on another. What matters is that the extracted components are equipped with a standard WSDL interface which converts the data in the requests to local data in the program and which converts the outputs of the program to data

in the requests. Creating such interfaces can be done automatically so there is no additional effort required here other than that of testing the interface. [26] The service itself will have been tested through years of productive use. The low costs and the short time in which recovered web services can be made available is the biggest advantage of this approach.

The major disadvantages are that

- the software is old and not easily comprehensible
- the conversion of the data from an external to an internal format reduces performance
- there may not be programmers available familiar with the legacy languages

An additional problem is that of maintaining data state from one transaction to another if the software was not originally designed to be reentrant. Reentrancy will have to be retrofitted into the software. Then the question of how to maintain the states of the many users comes up, but this is not a specific problem of recovered web services. It applies to them all.

## 4. Research Framework for Web Service Mining

The focus of the research proposed here is on how to recover web services from existing applications, also referred to as web service mining. Much of the business functionality of an organization has already been implemented at one time or the other. The problem is that it is not readily accessible. The functionality is buried in legacy software systems. [27]

To make this functionality available for reuse as web services it has to be extracted from the context in which it has been implemented and adapted to the technical requirements of a service-oriented architecture. This involves four different activities.

- discovering
- evaluating
- extracting and
- adapting

### 4.1 Discovering potential web services

In principle, every application function performed by the legacy code is a potential web service. Here one should note that a great portion of legacy code is dedicated to performing some technical function or to serving some obsolete data storage or data communication technology. Studies have shown that this amounts to almost 2/3s of the total code. That leaves only 1/3 of the code for achieving the application objectives. It is that code which is of interest for recovering. The problem is that the application focused code is highly intertwined with the technical code. Within one block of code, i.e.

method, module or procedure, there may be both statements for setting up a data display mask and statements for computing the value added tax. In searching through the code, the analyzer tool must recognize these statements which provide the business value. [28]

On the other hand not all business functions are correct. Many will have become obsolete over the years. So, not only must the application functions of the code be identified, they must also be checked for being currently of value. That leads to two research issues here

- 1) how to identify code performing an application function and
- 2) how to determine if that function is still of current value to the user

Both issues will require some form of rule based decision making. The important thing is that the user is able to adjust the rules to his needs, meaning that the analysis tools have to be highly customizable. They also have to be very fast, since they will be scanning through several million lines of code to identify which portions of that code are potential web services. So what is required here is actually a sophisticated search machine for processing source code. It will probably not be possible to select potential web services without the help of a human being. So there will also have to be some user interaction build into the tool to give the user the opportunity to intervene in the search and to make decisions which the tool cannot. Such interactive source mining tools is a major research topic.

The key to discovering potential web services in existing code has been described by this author in a previous paper on the recovery of business rules [29]. The approach is to identify the names of the essential data results and to trace how they are produced. This is achieved via an inverse data flow analysis. The data flow trace may pass through several methods or procedures in different classes or modules. It is necessary to identify all of them. An example is a credit rating. The essential result is the credit rate, but several modules or classes may be involved in producing that result. Therefore all of them must be combined to produce one single web service – compute credit rating. This problem is associated with the problem of impact analysis in software maintenance.

#### **4.2 Evaluating potential web services**

Another research issue in connection with the discovery of potential web services is the assessment of those code fragments selected as potential web services. Here the owner of the code must determine whether or not it is even worth extracting the code fragments from the systems in which they reside. This is a question of reusability.

Metrics have to be developed which will signal whether the code is reusable or not. The author has dealt with this problem before and published a paper on it [30]. The key metric is that of wrapability. A piece of software is wrapable if it can be readily separated from the code around it. It has few functional links, i.e. calls to functions in the surrounding code and it shares little data with the other procedures. Thus, one must count foreign calls and other branches outside of the code block in question as well as the non local data, i.e. data declared outside of that code block. These must then be related to the size of the code block or blocks in statements.

This may be a good starting point but it is not enough. There are also the questions of code quality and business value. The question is whether the code is good enough and valuable enough to migrate over into the new architecture. For that many of the existing quality metrics can be used. The business value must be measured in terms of how valuable the results produced by this particular piece of code are. There is very little research on this. Finally, it must also be calculated what it will cost to extract the code relative to the benefits of reuse. For this metrics are needed to measure the maintainability, testability, interoperability and reusability of the code as well as the business value of the functionality performed by that code.

The evaluation of potential web services is not a trivial matter and will require proven metrics to master. This is where the measurement community is called upon to provide a sound set of metrics to guide the decision whether to reuse or not.

#### **4.3 Extracting the web service code**

Once a code fragment has been identified as being a potential web service, the next step is to extract it from the system in which it is embedded. This can become a highly intricate task, especially when the code is not a separately compilable unit. Procedural modules may share global data with other modules in main storage. They may also call other modules. All of these dependencies must be capped in order for the procedural code to be extracted from its environment. Object-oriented code is generally easier to extract than procedural code, but there are also problems here. A particular class may inherit from higher level classes which one does not want to extract with it. The class may also invoke methods in foreign classes which return essential results. These dependencies have to be eliminated either by class flattening or by method inclusion. However none of these solutions are simple. Research is needed to determine the best means of extracting both procedures and classes.

A particularly difficult problem in extracting code from legacy systems is that of extracting features.

[31] Particularly in object-oriented software functionality is often dispersed through many classes contained in various components. A feature is a chain of distributed methods triggered by an event and resulting in a predefined output, such as the answer to a query or a computed result, for instance a price or a credit rating. To produce that result different methods in different classes must be executed in a given sequence. A proposed web service will most likely correspond to a feature. Extracting features from components poses a difficult challenge to the software research community. It is questionable whether it is possible to extract only those methods traversed by the feature, since those methods use class attributes which may affect other methods. On the other hand, extracting whole classes will result in very large web services containing a lot of code not relevant to the task of the web service at hand. Solving this problem, if it is at all solvable, will require a significant effort on the part of the research community.

#### 4.4 Adapting the web service code

The final research issue is that of adapting the extracted code units to be reused as web services. This entails supplying them with a WSDL interface. The input arguments which they received before either from a parameter list, a user interface mask, an input file or some other means of data input must be reassigned as arguments within the WSDL request. That means converting them from XML and moving them from the incoming SOAP message to the internal storage of the web service. The output results which they returned before as output masks, return values, output files, reports or other means of data output must be reassigned as results of the WSDL response. That implies moving them from the internal storage of the web service to the outgoing SOAP message and converting them into XML character format. [32]

All of these steps can be automated. In fact, code adaptation is the activity which lends itself best to automation. Yet, there is a still need for finding the best ways of doing this. Research is required on adaptation tool development.

## 5. Conclusion

This contribution has outlined different strategies for supplying web services to a service-oriented architecture. As pointed out, they can be bought, rented, borrowed, developed or recovered. There are advantages and disadvantages of each approach. The cheapest and quickest means of supplying web services, other than borrowing them from the open source community, is to recover them from the existing applications in the ownership of the user.

There is a pressing need for more research in this area of code recovery. First, techniques are required to identify code based on the results produced. Secondly, metrics are required to evaluate the reusability of the code identified. Thirdly, tools are needed to extract chains of interconnected code blocks, i.e. code features from the environment in which the code resides. Finally, tools are needed to automatically wrap the extracted code and adapt it as a web service. All of these steps are subject to automation, however, in order to automate them the optimal and most reliable means of performing these tasks must be investigated and different techniques compared. In this respect, the researches community can make a valuable contribution to the migration process.

### References:

- [1] Benamati, J./Lederer, A.: "Coping with rapid Changes in IT", *Comm. Of ACM*, Vol. 44, No. 8, August, 2001, p. 83
- [2] Glass, R.: "The Realities of Software Technology Payoffs", *Comm. Of ACM*, Vol. 42, No. 2, Feb. 1999, p. 74
- [3] Carr, N. "Software as a Commodity", in *Harvard Business Review*, No. 4, 2003, p. 51
- [4] Tockey, S.: *Return on Software*, Addison-Wesley, Boston, 2005, p. 319
- [5] Duchessi, P./Chengalur-Smith, I.: "Client/Server Benefits, Problems, best Practices", *Comm. Of ACM*, Vol. 41, No. 5, May 1998, p. 87
- [6] Sneed, H./Nyary, E.: "Downsizing Large Application Programs", *Journal of Software Maintenance*, Vol. 6, No. 5, Sept. 1994, p. 235
- [7] Terekhov, A./Verhoef, C.: "The Realities of Language Conversion", *IEEE Software*, Dec. 2000, p. 111
- [8] Seacord, R./Plakosh, D./Lewis, G.: *Modernizing Legacy Systems*, Addison-Wesley, Reading, 2003, p. 120
- [9] Garlan, D./Allen, R./Ockerbloom, J.: "Architectural Mismatch – Why reuse is so hard", *IEEE Software*, Nov. 1995, p. 26
- [10] Evaristo, R./Desouza, K./Hollister, K.: "Recentralization – The Pendulum swings back again", *Comm. of ACM*, Vol. 48, No. 2, Feb. 2005, p. 67
- [11] Bichler, M./Kwei-Jay, L.: "Service oriented Computing" *IEEE Computer*, March, 2006, p. 99
- [12] Bharati, P.: "India's IT Service Industry", *IEEE Computer*, Jan. 2005, p. 71
- [13] Andriole, P.: "Business Technology Strategies", *Comm. Of ACM*, Vol. 49, No. 5, May 2004, p. 85
- [14] Hammer, M./Champy, J.: *Reengineering the Corporation – A Manifest for Business Revolution*, Harpers Business, New York, 1993
- [15] Strassman, P.: "The Total Cost of Software Ownership", *IT Cutter Journal*, Vol. 11, No. 8, Aug. 1998, p.2

- [16] Krafzig, D./Banke, K./Schama, D.: Enterprise SOA, Coad Series, Prentice-Hall, Upper Saddle River, N.J., 2004, p. 6
- [17] Nguyen, H.Q./Johnson, B./Hackett, M.: Testing Applications on the Web, John Wiley & Sons, Indianapolis, 2003, p. 15
- [18] Tilley, S./Gerdes, J./Huang, S./Muller, H./Wong, K.: "On the Business Value and technical Challenges of adapting Web Services", Journal of Software Maintenance & Evolution, Vol. 16, No. 1, 2004, p. 31
- [19] Sneed, H.: "Generation of stateless Components", Proc. Of CSMR-2000, IEEE Computer Society Press, Zurich, March 2000, p. 183
- [20] Horowitz, E.: "Migrating Software to the World Wide Web", IEEE Software, May 1998, p. 18
- [21] Larson, G.: "Component-based Enterprise Frameworks", Comm. Of ACM, Vol. 43, No. 10, Oct. 2000, p. 25
- [22] Pak-Lok, P./Lau, A.: "The Present B2C Implementation Framework", Comm. Of ACM, Vol. 49, No. 2, Feb, 2006, p. 96
- [23] von Mayrhauser, A./Vans, A.: "Identification of Dynamic Comprehension Processes in large scale Maintenance", IEEE Trans. on S.E., Vol. 22, No. 6, June, 1996, p. 424
- [24] Bishop, J./Horspool, N.: "Cross-Platform Development – Software that lasts", IEEE Computer, Oct. 2006, p. 26
- [25] Aversano, L./Canfora, G./deLucia, A.: "Migrating Legacy System to the Web", in Proc. of CSMR-2001, IEEE Computer Society Press, Lisabon, March 2001, p. 148
- [26] Sneed, H.: "Wrapping Legacy COBOL Programs behind an XML Interface", Proc. Of WCRE-2001, IEEE Computer Society Press, Stuttgart, Oct. 2001, p. 189
- [27] Canfora, G./Fasolino, H./Frattolillo, G.: "Migrating Interactive Legacy System to Web Services", Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 23
- [28] Sneed, H.: "Integrating legacy Software into a Service oriented Architecture", in Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 3
- [29] Sneed, H./ Erdoes, K.: "Extracting Business Rules from Source Code", Proc. of IWPC-96, IEEE Computer Society Press, Berlin, March, 1996, p. 240
- [30] Sneed, H.: "Measuring Reusability of Legacy Software" in Software Process, Volume 4, Issue 1, March, 1998, p. 43
- [31] Greevy, O./Ducasse, S./Girba, T.: "Analyzing Software Evolution through Feature Views", Journal of Software Maintenance & Evolution, Vol. 18, No. 6, Dec. 2006, p. 425
- [32] Bodhuin, T./Guardabascio, E./Tortorella, M.: "Migrating COBOL Systems to the WEB", WCRE-2002, IEEE Computer Society Press, Richmond, Nov. 2002, p. 329