Assessment of a Framework to Compare Software Development Methodologies

Riaan Klopper
Department of Computer Science
University of Pretoria
Pretoria, South Africa, 0002
riaank@strate.co.za

Stefan Gruner
Department of Computer Science
University of Pretoria
Pretoria, South Africa, 0002
sgruner@cs.up.ac.za

Derrick G. Kourie
Department of Computer Science
University of Pretoria
Pretoria, South Africa, 0002
dkourie@cs.up.ac.za

ABSTRACT

A decision-supporting framework was applied in a pilot study to assist in the decision making about what software development methodology to use at a software engineering company. This paper critically assesses this decision making process and framework that was used at that company to decide on an appropriate software methodology for the analysis and design of business processes and software systems.

Keywords

Aris, Business process analysis and design, MDA, RUP, Software development methodology, UML, URDAD.

ACM Categories [Subject Descriptors]

D.2.1 [Software Engineering, Requirements]; D.2.9 [Software Engineering, Management]; K.6.3 [Software Process Management].

1. INTRODUCTION

In a memorandum published almost exactly ten years ago to the date of this conference, Gregor Snelting has strongly criticised the inflationary proliferation of "new" software engineering concepts and methodologies, especially from the academic community [8]. He stated that concepts and methodologies are merely produced for the sake of publication, but would never be subject to any attempt of empirical evaluation. Taking Snelting seriously, we would like to be able to make a well-informed decision about which of the many available software methodologies is useful – and why– for a particular software engineering business in a particular situation.

Making a decision, from an organisational perspective on what software development methodology to use, is no small task. Numerous users, service providers, and stakeholders are affected by it, and therefore need to take part in the process to decide on the appropriate software methodology. These groups all represent different views and needs, resulting in a decision making process that is quite often underpinned and affected by strong emotions. A need for a more structured and rational approach to aid this decision making process is apparent. The aim of this paper is to critically assess the framework and process that was used in a pilot study conducted at *Strate Ltd*., South-Africa, to decide on an appropriate software development methodology for this company.

Each step that was followed, and each activity that was executed will be explained, and assessed. The assessment of this process and framework will therefore run throughout the paper, and be integrated into the entire text which aims to explain the process. The results will be summarised at the end of the paper. Special emphasis will be placed on the question which ones of the many criteria (or parameters) can be regarded as *decisive* as far as the selection between several similar alternatives is concerned. A pilot study can hint "qualitatively" at those parameters (or criteria) which seem to be more significant and more decisive than others, and which should therefore be used as the main guidelines in the further development of a refined decision support framework (which might be even tool-supported in the not-too-far future).

Thus, this paper constitutes an example of empirical research in the spirit of Snelting's memorandum [8], and it also employs qualitative research principles. It presents largely an interpretivist view and can also be regarded as participatory research, as one of the authors was part of the process and not an objective bystander. The results of the research, however, are still considered to be "objective" in a sense of inter-subjectivity, last but not least due to the considerable number of experts involved in the pilot study as commentators or interviewees, (see acknowledgments below).

The framework that was chosen to assist in this comparison exercise is the framework that was put forward by Avison and Fitzgerald [1]. The prescribed framework in itself was not enough to do the comparison with, and as such it was supplemented by further processes that will be explained in this paper. The scope of this paper therefore includes the assessment of both the framework as prescribed by Avison and Fitzgerald, and the process that was used to apply this framework. The framework and the process can be regarded as a (however not yet tool-supported) *Decision Support System*, aiding management in the decision making process. It can only produce recommendations, not decision. The decision is still the responsibility of the individual decision makers of an organisation.

It is important to not confuse the research approach of this paper with the approach of the decision support process that is under investigation.

SAICSIT 2007, 2 - 3 October 2007, Fish River Sun, Sunshine Coast, South Africa

2. SOFTWARE METHODOLOGY

It is important to understand the basic concepts of a software methodology in order to evaluate a process and framework that assesses methodologies. This will enable the reader to put the process and framework that is under investigation in a better context, and aid in the understanding of the presented results.

The BCS Information Systems Analysis and Design Working Group defined a software methodology as "recommended collection of philosophies, phases, procedures, rules, techniques, tool, documentation, management, and training for developers of information systems" [1]. According to Berard [2], a good software methodology is a methodology which:

"can be described quantitatively, as well as qualitatively,

can be used repeatedly, each time achieving similar results,

can be taught to others within a reasonable timeframe,

can be applied by others with a reasonable level of success,

achieve significantly, and consistently, better results than either other techniques, or an ad hoc approach, and

are applicable in a relatively large percentage of cases."

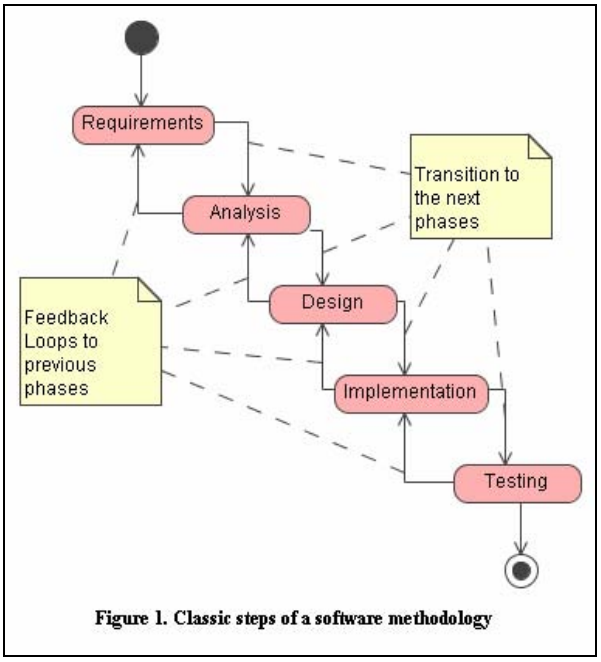

Figure 1. Classic steps of a software methodology

In its simplest form, a methodology to develop software usually has the following steps [11].

Requirements gathering

Analysis

Design

Implementation

Testing

As depicted in **Figure 1**, each phase loops back to the previous phase, in order to make corrections based on new information and a better understanding. This simplistic view of a software methodology is known as the *Waterfall model*. It is not recommended to use this as an actual software methodology, as it has many drawbacks. It is still very useful, though, to use it as a benchmark to help conceptualise other methodologies with [11].

There are many different variations of this simplistic representation of a software methodology. Many software methodologies take on different characteristics in order to address the inherent difficulties of software development [11]. It is the choosing between these variations that is the topic of this paper. In essence, however, the principles of all these methodologies still stay "the same" – which makes the choice process a difficult task.

Construction of quality software is not an easy activity. It includes various stakeholders and participants, and involves a significant amount of complex code, data, etc. It is therefore very risky to assume that one can construct quality software without any kind of a process to offer some guidance in crucial situation. However, according to our experience, there exists a significant amount of practitioners who do not see the need for a more rigorous approach to develop software.

Against the methodology sceptics, Fitzgerald presents several arguments in favour of organisations using software development methodologies [4]:

Methodologies help to cope with the complexity of the software development process.

Methodologies reduce risks and uncertainty by rendering the development tasks more transparent and visible.

Methodologies may provide a framework for the application of techniques and resources at appropriate times during the development process.

Standardisation of the development process is available. This aids the interchange ability of software developers. This is even more important when development work is outsourced.

Certification (ISO, CMM, TMM, etc.) becomes possible for organisations.

Some governments and institutions, especially in safety-critical domains (e.g. military, air traffic control) require that certain methodologies be used, as it increases the quality of software and also offers a certain degree of legal protection to all parties involved: for example, any software produced for German national or regional authorities must comply with the ***V*** development model.

Given the need for a software methodology, the challenge of choosing which software methodology to adopt comes to light. It is with that question in mind that this paper is put forward as a pilot study, with the intent of enhancing it further, and thereby adding value to the industry at large.

There are various different notations available for modeling software systems. Part of the organisational requirements was to have a standardised, internationally accepted and maintained notation that can model both technical systems and business processes. The well-known Unified Modelling Language (UML) has been chosen as the underlying notation in the context of our methodology studies, and we assume the reader to be familiar with it [10].

3. PILOT STUDY: OVERVIEW

The high level (meta) process that was followed to facilitate the decision making process for selecting a software methodology is outlined in **Figure 2** below. As mentioned above, this serves as a Decision Support System that facilitates the decision making process. The output is a set of recommendations (not the actual decision), as there are usually different subjective (opinion-based) considerations that have to be balanced in further negotiations between the various stake-holders (such as: clients, managers, programmers) of a software engineering project or company.

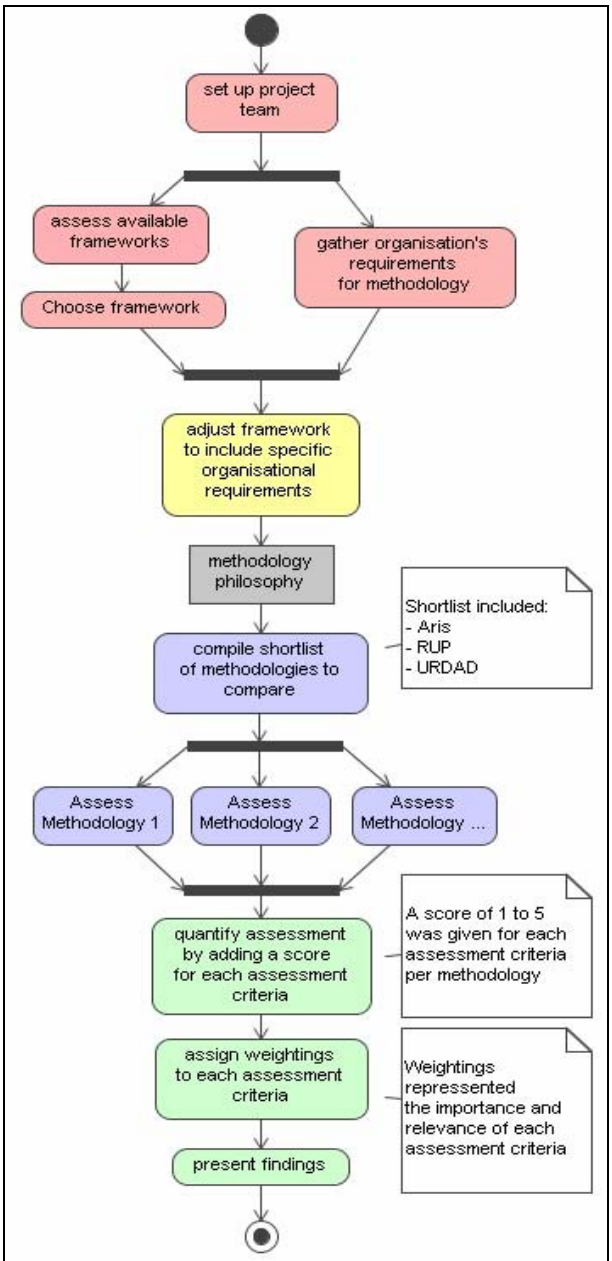

Figure 2. Decision making Process followed

We started by identifying an *appropriate team* to conduct the comparison exercise. Previous experience in applying software methodologies was a prerequisite for team members to participate in this process. The reason for this is that the theory of how a software methodology should be applied is quite often different from the way that it is actually applied (see below). Identifying an appropriate team is usually a constraint for an organisation who wishes to conduct its own process for selecting a software methodology. This process is reliant on having sufficient expertise in this matter. If the organisation does not have the relevant experience in-house, it has the option to outsource this responsibility to an external vendor. That comes with a host of other complications, though, which is outside the scope of this paper.

Once the team was identified, different *frameworks* were investigated to guide the decision making process. By "framework" we basically mean a meta-method according to which a suitable development method for the given organisation in its given situation shall be found. An appropriate framework was chosen, that supported the organisational requirements for the assessment exercise. One of these framework *requirements* was the ability to *quantify* (however crudely) the results of the assessment exercise. This meta-requirement is mentioned here as it turned out to have an important bearing not only on the chosen framework itself but also on the way in which the pilot study was conducted by its participants.

Next, the *organisational requirements* for a software development methodology were gathered. At a very high-level, these organisational requirements included the following:

- A software methodology that would *facilitate the communication* and understanding between the business and IT divisions,
- A software methodology that would create and maintain a *library of business processes* at an enterprise level, which can be reused across different strategic projects,
- A software methodology that would *maintain a model* of a software system, from where understandable business documentation, as well as technical documentation can be generated,
- A *common notation* that can be used for business and technology minded people alike,
- An environment where different stakeholders of a project can *collaborate* to build and maintain a software model,
- A software methodology that would facilitate business process *re-engineering,*
- A software methodology that can *integrate with existing processes* such as the project management methodology, quality assurance, change control, etc., and
- A *standardised process* that can be repeated for different projects and still yield "the same" results.

Once the organisation's requirements were gathered, and the framework for comparing methodologies was chosen, the framework was fine-tuned to include the organisational requirements for the software methodology.

The framework was then used to select a shortlist of *candidate-methodologies* to be compared. The software methodology's underlying principles were used to achieve this, as it was possible to determine if an organisation's requirements would be met by these underlying principles. This ensured that similar methodologies are selected and compared against each other, comparing "apples with apples" instead of "apples with pears", thus resulting in a sensible planning exercise. Aris [9], RUP [11], and URDAD [12] were chosen as candidate methodologies. Another factor that played a role in these software methodologies being chosen was the practical experience that the business analysts already had in working with them.

The framework for comparing software methodologies in our pilot study contained an initial set of different *assessment criteria* (parameters) to apply to each methodology. An analysis of each software methodology was conducted using these assessment criteria, and a score was allocated per assessment criteria for each methodology. *Weights* were assigned to each assessment criteria to make it more relevant to the specific organisational requirements that were gathered previously. As mentioned above, an important purpose of the pilot study was to reflect on the appropriateness of the initially chosen assessment parameter, especially as far as their discriminating (distinguishing) capacity is concerned –in simple words: how useful are these parameters– such that similar assessment exercises in the future can be based on a further refined set of assessment criteria.

Finally, the results of the choice process (see below) were presented to the company's relevant stake-holders in an in-house conference.

4. PROCESS REQUIREMENTS

The process for selecting and comparing the various methodologies also needed to address some pre-defined organisational requirements. A successful project is usually defined as a project that delivers a product which meets the original requirements. Similarly, to evaluate the efficiency and success of this process, the results of it need to be compared against the original requirements.

The organisational requirements that were identified for a process and framework to compare software development methodologies are, a framework that:

not only supports the comparison of software development methodologies, but also aids in the preliminary selection of the various methodologies that will be compared,

provides meaningful and relevant results (comparing "apples with apples", not "apples with pears"),

provides results that are quantifiable,

is repeatable,

facilitates the involvement of multiple stakeholders who has an interest in the software methodology being chosen,

provide results that are useable enough to aid management in making informed decisions, and also

considers the organisational requirements behind the to-be-chosen software development methodology.

5. ORGANISATIONAL BACKGROUND

Strate Ltd. is the central securities depository of South Africa and employs approximately 140 members of staff. The company is responsible for the clearing, settlement, and corporate action activities for the majority of the electronic trades conducted on the Johannesburg Securities Exchange (JSE), YieldX, and the Bonds Exchange of South Africa (BESA). It has direct links into the South African Reserve Bank (SARB) and the SWIFT payment network to facilitate the transfer of money in central bank funds. Software technology is therefore extremely important to all of the operations of the organisation, and failure of such constitutes a systemic risk to the entire South African Financial Market.

The company consists of mainly three areas:

Multiple business divisions, the most important being the Clearing & Settlements and the Issuer & Asset Services Division,

IT division, and

Enterprise Development division.

The Enterprise Development division is responsible for management and implementation of technology intensive projects. The Business Analysis Department is part of the Enterprise Development Division, and interfaces with the Business Divisions as well as the IT Division.

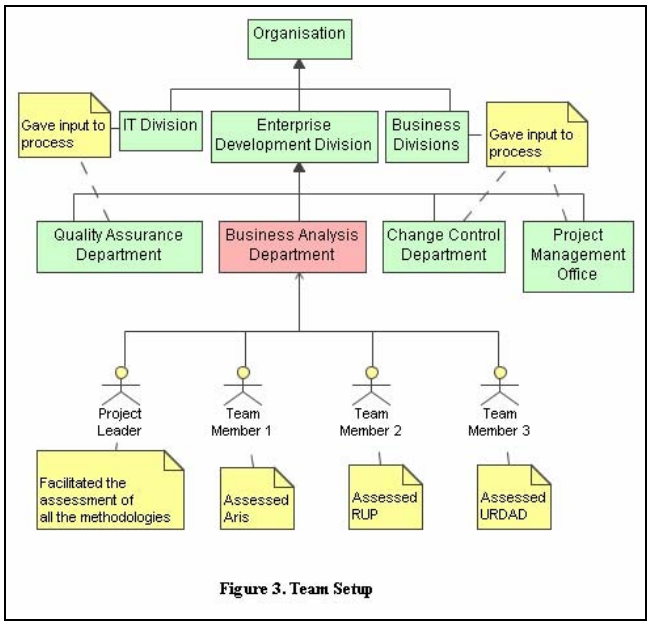

Figure 3. Team Setup

As a result of the intent of a software methodology to be far reaching and bridging gaps, *many different role players* needed to be involved in the decision making process pertaining for a software methodology. The majority of the process was, however, still conducted by the Business Analysts, who possessed the working experience in various methodologies. The project leader, together with three other Business Analysts collaborated to facilitate the processes and make an informed decision on what software methodology would be the most appropriate for the organisation.

Even though the business analysts were involved in driving the process, it was endeavoured to involve the opinions of as many as possible business and IT resources. This task was challenging, and it was felt that better participation of the business and IT divisions in the initial phases of the process would have improved the end results, and increasing the organisational understanding of what the chosen software methodology would be expected to achieve.

Another challenge was the fact that only four business analysts were involved in the process. The statistical accuracy would have been greatly improved if more resources were available for this, as personality influences may have been averaged out. It is therefore recommended more than six resources are involved in driving the process when three methodologies are evaluated. This is purely a recommendation based on the experiences gained, and further studies could help define these variables more accurately.

We believe that the process was sufficiently fair in giving each analyst the opportunity to voice their concerns, and having one person being the common denominator to ensure unified understanding and application of the process.

6. ASSESSMENT OF FRAMEWORKS

A number of frameworks were identified as possible means of comparing methodologies. The most prevalent of these was a framework put forward by Avison and Fitzgerald [1]. The other frameworks that were considered are mentioned here as well.

6.1 Bjorn-Anderson's Framework

Bjorn-Anderson suggested a much broader range of issues to consider when choosing a software development methodology than Avison and Fitzgerald [1]. Criteria relating to values and society are used to assist in the evaluation. Some of them include:

What research paradigms form the foundation of the methodology?

What are the underlying value systems?

In what context is a methodology useful?

To what extent is modification possible?

Does communication and documentation consider the user's dialect?

Does transferability exist?

Is the societal environment dealt with?

Is user participation encouraged?

The above list is broad and rather subjective. It is also stated that it makes some assumptions, such as if user participation is really desired [1].

6.2 NIMSAD

Normative Information Model-based Systems Analysis and Design (NIMSAD) is based on the models and epistemology of systems thinking and mostly evaluates a methodology against these criteria. The evaluation consists of three elements [1]:

The 'problem situation' (the methodology context).

The intended problem solver (the methodology user)

The problem solving process (the methodology)

This framework aims to evaluate a methodology during three stages. First, before the methodology is adopted, second during its use, and third after an assessment of the success of the methodology. It therefore takes into account *organisational learning*.

6.3 Davis's Framework

Davis [1] advises the contingency approach, by selecting of an approach as part of the framework or methodology itself. He offers guidelines to select an appropriate approach to the determination of requirements, rather to the selection of a methodology itself. He suggests measuring the level of uncertainty of a system, by taking into account [1]:

System complexity,

The state or flux of the system,

The user component of the system, and

The level of skills and experience of the analysts.

An approach to choose and compare a methodology is therefore chosen depending on the specific situation. If there are, for example, low levels of uncertainty, interviewing users to gather requirements might be appropriate. For higher levels of uncertainty, a prototype could, for example, be used.

6.4 Avison and Taylor's Framework

Avison and Taylor [1] identify five different classes of situation, and then suggest appropriate approaches to address these situations:

Well structured problems with clear requirements: A traditional *SDLC* might be appropriate.

Well structured problems with unclear requirements: A data, *process modeling*, or *prototype* approach may be appropriate.

Unstructured problem situation with unclear objectives: A "soft" system approach may be appropriate.

High user interaction system: A people focused approach such, as *Ethics*, may be appropriate.

Very unclear situations: A contingency approach, such as *Multiview*, may be appropriate.

This framework is useful to choose what kind of methodology should be used, but could prove difficult as a tool to compare different methodologies.

6.5 Comparison

These different frameworks all have their respective strengths and weaknesses. A concern is that they all provide very subjective unspecific assessment criteria. Another challenge is that the number of assessment criteria that is available in the above frameworks might not be enough. When the analysis of each methodology is exploratory in nature, there is a need to have a significant number of assessment criteria, as the more criteria that are available the better supported the decision would be. The chosen framework will be presented next.

7. THE CHOSEN FRAMEWORK

The framework as suggested by Avison and Fitzgerald [1] was chosen to aid the decision making process. The main reason for choosing this framework was the precise assessment criteria that it provides to compare the different methodologies against, in contrast to the other frameworks what were regarded as too imprecise and subjective.

Each assessment criterion that is supplied in Avison and Fitzgerald's framework is mentioned below, as well as a short explanation.

The Software Methodology's Underlying Principles plays an important role in understanding what a particular methodology is about. According to Avison and Fitzgerald [1], the underlying principles of a methodology underscore all other aspects. By looking at the underlying principles, one can distinguish a "method" from a "methodology". The choice of the areas covered by the methodology, the systems, data or people orientation, the bias or otherwise toward a pure IT solution and other aspects are made based on the underlying principles of the methodology. These underlying principles may be explicit or implicit.

As a guide to the underlying principles the four factors of **paradigm**, **objectives**, **domains**, and **targets** are highlighted below [1].

There are two **paradigms** of relevance [1]. The first is the science paradigm (which has characterised most of the scientific developments of recent times), and the second is the systems paradigm (which is characterised by a holistic approach).

According to Kuhn, as quoted by Avison and Fitzgerald [1], a paradigm is a specific way of thinking about problems, encompassing a set of achievements which are acknowledged as the foundation of further practice. A paradigm is usually subject free, and is generic enough to be applied to a number of problems regardless of the content. An example of a paradigm is object orientation, or iterative driven approaches, etc., as further discussed in [6].

The **objectives** of a software methodology define what the methodology is trying to achieve. It can, for example, state that the methodology is only concerned with system development, while the objectives of other methodologies might be to take a wider view into account, for example, to re engineer the organisation's business processes. This is not the same as the scope of a methodology that will be discussed later, which is related to the exclusion and inclusion of certain steps of the development life cycle. The objectives are focused on the boundaries of concern that the methodology should address.

The **domain** refers to the domain of situations that a software methodology addresses. Some methodologies, for example, only concentrates on a narrow view, and does not take into account a broader view, such as the strategic requirements of the organisation. The domain criterion is therefore concerned with aligning business and IT goals, and how well the methodology supports that.

An example may be to have requirements traceability from requirements to the system realisation, and as such, one could asses if the solution meets the needs of the organisation.

The **target** refers to the applicability of the methodology. The question to be answered is to what situation, culture, or organisation the methodology is targeting.

The software development methodology's "philosophy" was used to decide on a short list of methodologies to include in the comparison exercise. This was achieved by taking into account the organisational requirements that were gathered, and only choosing those methodologies whose "philosophies" supported these requirements.

Another important factor that was considered was to ensure that only software development methodologies were selected that supports the organisational culture. This saves a significant amount of time, by only considering methodologies that could be practically implemented if selected.

The Separation of Logical Design implies that there should be a separation of requirements, and how it is implemented, i.e. what versus how [1]. This is also outlined by the MDA (Model Driven Architecture), maintained by the OMG (Object Management Group), which specifies that analysis and design should produce a PIM (Platform Independent Model). The separation of logical and physical design, though, does not imply that requirements should not be traceable to design and implementation [12].

Rules support the notion that methodologies should provide formal guidelines to cover phases, tasks, and deliverables, and their ordering, techniques and tools, documentation and development aids, and guidelines for estimating time and resource requirements [1].

The Model is the basis of the software methodology's view of the world. It is an abstraction and a representation of the important factors of the information system or business process. The model works on a number of difference levels [1]:

Means of communication;

A way of capturing the essence of a problem or design, such that it can be translated or mapped onto another form (e.g. implementation) without loss of detail; and

It is a representation which provides insight into the problem area of concern.

Models can be:

Verbal;

Analytical or mathematical;

Iconic, pictorial, or schematic; or

Simulation.

Most information systems methodologies are of the iconic, pictorial, or schematic type.

The Techniques and Tools referred to here are the various strategies and technologies used to support the methodology. An example might be that RUP uses UML and "Rational Rose".

The Scope of a software methodology refers to the amount of activities that it covers in the software development project life cycle. A methodology should ideally cover the entire systems development process from strategy to maintenance [1].

The Outputs of a methodology are measured at every step of the process, not just at the end. Different outputs can be delivered at different points. They key factor here is that the methodology

should be able to produce a certain artefact at a point in time in order to support a specific process from another discipline. An example might be that the methodology should be able to produce estimates early in the life cycle, in order to integrate with the project management methodology.

The Practice of a methodology refers to the gap between the intended use of a methodology, i.e. the theory, and the actual application in practice. According to Avison and Fitzgerald [1], the practice can also refer to the degree to which the methodology can be, and is altered or interpreted by the users according to the requirements of the particular situation. It can be viewed in terms of the following: The *background* refers to the origin and intended use of a methodology, such as academic or commercial, the *user base* refers to the numbers and the types of users, and the *participants* refer to the participation of various role players, such as users and analysts.

The Understanding of the Information Resource refers to the ability of a methodology to ensure effective utilization of an organisation's information resources, such as existing business process and data [1]. This implies the knowledge share and reuse.

Documentation Standards refers to the ability of a methodology to output documentation according to agreed standards that are understandable by business and technical users [1]. Internationally accepted and standardized notation such as UML should be used. Further more, a methodology should be able to proved output for a variety of audiences, such as high level documentation for business executives, more detailed business processes for operational staff, and technical documentation of IT resources.

The Validity of Design dictates that there should be a means of checking for inconsistencies, inaccuracies, and incompleteness of the deliverables of a methodology [1].

Early change refers to the requirement that any changes to a system design should be identifiable as early as possible in the life cycle [1].

Inter-stage Communication supports the notion that the full extent of work carried out should be communicable to other stages, with each stage being consistent, complete, and usable [1].

Planning and Control is a management requirement for software development methodologies. Careful monitoring is required, as well as the support of development in a planned and controlled manner in order to contain costs and time scales [1]. The methodology should therefore be integrate-able with other processes, such as the project management and quality assurance processes.

Performance Evaluation refers to the ability of a software methodology to support a means of evaluating the performance of operational application developed using it (or business processes implemented).

Increased Productivity should be visible for users of the methodology, such as analysts, as well as for stakeholders of the project that was undertaken using the methodology [1].

Improved Quality for a software methodology is measurable in terms of the improvement of the quality of analysis, design, and programming products, and hence the overall quality of the information system [1]. It should also be measurable in terms of the quality of a redesigned business process.

Visibility of the Product requires that a methodology should maintain the visibility of the emerging and evolving information system as it evolves [1]. This assists in more effective project management and risk management activities.

The Teach-ability of a software methodology refers to how easy it can be taught to others. Users as well as technologists should understand the various techniques in a methodology in order to verify analysis and design work, and train others to use it [1].

The Information System Boundary refers to the ability of a software methodology to allow definition of the areas of the organisation to be covered by the information system. These may not all be areas of computerisation [1].

Designing for Change requires that the logical and physical designs should be easily modified [1]. This will ensure that the ability to effectively maintain a software project after deliver is greatly increased.

Effective Communication should be provided between analysts, IT, and users [1].

Simplicity refers to the ease of use of the software methodology [1].

Ongoing Relevance refers to the saleability of software methodology. It should be capable of being extended so that new techniques and tools can be incorporated as they are developed, but still maintain overall consistency and framework [1].

Automated Development Aids refer to software and tools that can be used with the software methodology. This should be practiced wherever possible, as they can enhance accuracy and productivity [1].

The Consideration of User Goals and Objectives of potential users of a system should be noted, so that when an information system is designed it can be made to satisfy these users and assist them in meeting goals and objectives [1]. An example could be the use of use-cases to capture a user's goals, and have requirements traceability through to realisation.

Integration of Technical and Non-Technical Systems refers to the ability of a software methodology to not only address the technical and non-technical aspects of a system, but should make provision for their integration [1].

Scan for Opportunity refers to the ability of a methodology to enable the system to be thought about in new ways. Rather than being viewed as simply a solution to existing problems it should be seen as an opportunity to address new areas and challenges [1].

Product and Cost refers to what purchasers actually get for their money. This might include software, written documentation, telephone support, consulting, and training.

These assessment criteria are more specific than that of the other frameworks mentioned earlier, and therein lies their usefulness. Assessing methodologies in an exploratory way is greatly supported by having more criteria as opposed to less.

8. ORGANISATION REQUIREMENTS

Once the assessment criteria above were understood, it was important to ensure that they also fully represent the organisational requirements for a software development methodology.

An exercise was conducted to make sure that each of the organisational requirements that were mentioned earlier was represented by at least one of the assessment criteria above. It was also investigated whether some assessment criteria contradicted an organisational requirement. None were found.

By taking into account the organisational requirements of a software methodology, the process ensures that the framework is customisable for any company of any size, in any industry.

9. METHODOLOGY CANDIDATES

By matching the organisational requirements for a methodology, to the underlying principles of available methodologies, a shortlist was chosen. The following methodologies were chosen for the comparison exercise:

Aris (ARchitecture for integrated Information Systems)

RUP (Rational Unified Process)

URDAD (Use-case Responsibility Driven Analysis and Design)

All of the above methodologies have certain best practice commonalities that will ensure that they are indeed comparable, and that the result of the comparison is quantifiable. One such commonality is that they are all *iterative and incremental* approaches, and they all utilise the UML (Unified Modeling Language) as specified and maintained by the OMG (Object Management Group).

When selecting a methodology, it is important to not only select it based on its theoretical promises. Many methodologies only display academic concepts, without accompanying empirical evidence to support its claims. Some authors, such as Snelting, have identified this as a problem. As mentioned above he had published a memorandum to software professionals for more stringent methodological standards and as well as their empirical validation in software technology [8]. He also argued that predictions need to be *falsifiable*, and as such, if a methodology predicts a certain outcome, that outcome needs to be measurable in order to test the theory, as further discussed in [6]. This is one of the driving factors for only choosing methodologies that have been implemented in practice, in order to have empirical evidence available to assess them with.

9.1 Aris

The Architecture of Integrated Information Systems (Aris) claims to be the benchmark for enterprise-wide Business Process Management. It is owned by IDS-Scheer, an international IT process company [9].

Aris is a methodology that comes bundled with commercial software. This means that Aris as a methodology is obtainable if one purchases licences for the software.

9.2 RUP

The Rational Unified Process (RUP) is a methodology that is owned and maintained by IBM. It is probably the most widely used methodology; its main three contributors were Jacobson, Booch and Rumbaugh [11].

RUP is bundled with commercial software called *Rational Rose*. The RUP methodology is available in its complete format when one purchases licences for the software.

9.3 URDAD

The Use case Responsibility Driven Analysis and Design (URDAD) methodology is a novel methodology that aims to bring together the most widely used best practices. It consists of a simple algorithm that produces a Platform Independent Model (PIM) as defined by the Model Driven Architecture (MDA) [12].

It was developed and is maintained by a South African based IT training and consulting firm called Solms Training Consulting and Development (STCD). It is freely obtainable under the Public License Agreement.

URDAD does not come bundled with commercial software. It was our approach to use Magic Draw UML in combination with URDAD, as it is an UML modeling tool that supported the URDAD methodology quite well.

10. ASSESSMENT AND CHOICE

It was attempted to select methodologies that are similar, and by doing that, increase the accuracy of– and thereby, also the difficulty of– our comparison exercise.

The organisational requirements were used to select the shortlist of methodologies, by investigating their underlying principles. A challenge was that most software companies guard their intellectual capital very closely. Unfortunately, it makes it difficult to gain access to the underlying principles of the methodology, without purchasing the methodology. One is therefore reliant on previous assessments and personal experiences of the methodologies.

The result of this was that we had to choose methodologies about which we had information and experience, to be part of the comparison exercise. Unfortunately it is difficult to know which methodologies were missed as a result of this constraint.

10.1 Assessment Process

Each software methodology on the shortlist was assessed per assessment criteria as per Avison and Fitzgerald's framework [1]. Each analyst was assigned to a software methodology based on the level of experience on that methodology. This was possible, as the experience of the analysts were taken into account when the respective software methodologies were chosen.

Because of resource constraints, there was one analyst assessing each of the three software methodologies. The team leader of the project was involved in the assessment of all three methodologies in order to ensure a unified understanding of each assessment criteria.

10.2 Collaborative Scoring

Once the different software methodologies were assessed using the assessment criteria of the framework, the results were preliminarily quantified. However, this quantification was still largely based on subjective "intuition", as it is typical for an initial pilot study. (Subsequent studies are expected to be more accurate, based on the insights gained from the pilot study.)

This exercise was conducted as a collaborative effort. An in-house workshop was arranged and each analyst had to present their findings for each assessment criteria. The score for each methodology per assessment criteria was then discussed and agreed on.

A numeric score of between 1 and 5 was given for each criterion, with the following interpretation:

Very bad: "Does not meet this requirement of this criterion",

Bad: "Hardly meets the requirement of this criterion",

Average: "Modestly meets the requirement of this criterion",

Good: "Meets the requirement of this criterion very well",

Very good: "Meets the given requirement exceptionally well"

There was a concern that an even scale should have been used in order to avoid the middle ground option, i.e. choosing "average" too much. This was not needed in this instance, as most scores ranged between 4 and 5 (see **Table 1** below). The reason for this was that the shortlist only contained strong industry standard methodologies of similar nature that naturally scored well on most of the assessment criteria. For a more generic framework, even scales should be considered in the future.

Another concern was the *risk of double scoring*. Some assessment criteria might be very similar to others. In principle, all the assessment criteria are different, and as such, if one interpret them as the same, it might constitute a lack of understanding of that particular assessment criterion and it should be investigated further. This risk can further be reduced by adjusting the scores for similar assessment criteria by using weightings (see below).

10.3 Assignment of Weights

In order to make the method more relevant to the organisation in question, the organisation's requirements for a software development methodology also had to be integrated into the quantified results. This was done to address the organisational requirement for the process of choosing a software methodology to be meaningful a relevant to the organisation.

This was achieved by attaching weights to each assessment criterion, with assessment criterion that are more relevant to the organisation receiving a larger weight, and the criterion that were less relevant to the organisation, receiving a smaller weight. The decisions about the weights-values were based on the organisational requirements on software engineering methodology as described and explained above.

The concern was that the weights should have been defined *before* the collaborative scoring exercise. We believed, however, that an adequate understanding of the assessment criterion and the organisational requirements for the software process methodology was only at a sufficient level late in the process, making the weighting decisions more meaningful at this point. In this way we also avoided the problem of whom to select as a privileged person to define the weights before the assessment sheet was handed out to the members of the assessment team.

The following weightings were assigned to each assessment criteria's based in the relevance to the organisation:

0.5 – "Low" relevance

1.0 – "Normal" relevance

1.5 – "High" relevance

11. RESULTS

Our results are twofold. Firstly, a better understanding of the respective methodologies was gained in order to aid the decision making process. Secondly, some statistical outputs were generated to aid in the decision making process. The results are displayed in **Table 1**, which shows the assessment criteria, as well as the normal and adjusted scores for each methodology. There were some assessment criterion for which the respective methodologies scored equally. In order to simplify the results, these are aggregated in the line item "Equal Scoring Criterion".
It seemed that URDAD scored the highest, and as such, promised a higher benefit realisation. It should also be noted that the actual scores are very similar, and as such, one can question its relevance. As was stated previously, the scores are only intended to aid the decision making process, and is not a final decision in itself.
For example, if cost is a major consideration for an organisation when choosing a software development methodology, then this data would be able to approximate a cost benefit ratio. In our cost analysis, initial investment costs were estimated per methodology, and this was used to assign the scores for the "product and cost" assessment criterion. If "Product and Cost" is removed from **Table 1**, a cost benefit ratio can be calculated for each methodology: see **Table 2**. This can in fact be seen as the amount of Rands spend per benefit point, and as such, the lower the ratio, the better the investment: see **Table 2**.

Table 1. Methodology Scores

Assessment Criteria	Weighting	Aris		RUP		URDAD	
		Normal	Adjusted	Normal	Adjusted	Normal	Adjusted
Rules	1	4	4	3	3	4	4
Model	1	3	3	3	3	5	5
Techniques and Tools	1	5	5	5	5	5	5
Scope	0.5	3	1.5	5	2.5	3	1.5
Documentation Standards	1	3	3	4	4	4	4
Planning and Control	1.5	4	6	3	4.5	4	6
Improved Quality	1.5	4	6	4	6	4	6
Teachable	1.5	3	4.5	3	4.5	4	6
Information System Boundary	1	3	3	4	4	5	5
Designing for Change	1.5	4	6	4	6	4	6
Effective Communication	1.5	4	6	4	6	4	6
Simplicity	1.5	3	4.5	3	4.5	5	7.5
Automated Development Aids	0.5	4	2	4	2	4	2
Consideration of User Goals and Objectives	1.5	4	6	4	6	4	6
Scan for Opportunity	0.5	4	2	4	2	4	2
Product and Cost	1.5	1	1.5	2	3	5	7.5
Equal Scoring Criterion		49	49	49	49	49	49
		105	**113**	**108**	**115**	**117**	**128.5**

Table 2. Cost-Benefit Estimation

	Aris	RUP	URDAD
Adjusted Score excluding Product and Cost	111.5	112	121
Initial Investment Cost	1,100,000.00	800,000.00	70,000.00
Cost Benefit Ratio	9,865.47	7,142.86	578.51

It was established from all of the results that the URDAD methodology should be the chosen methodology for the organisation, but the decision still remains a subjective exercise. These results were presented to the decision makers, accompanied by the recommendations of the business analysis team. The decision was then made by the decision makers, based on both the

qualitative and quantitative data, that the organisation would adopt the URDAD methodology.

12. SUMMARY AND FUTURE WORK

The contribution of this paper is an assessment of a framework to compare software development methodologies. The framework was presented as well as the requirements that such a framework should achieve. The practical experiences of the facilitators have been integrated into the explanation of the process to apply this framework.

As was stated previously, the original requirements for a framework to compare software methodologies are a framework that:

not only supports the comparison of software development methodologies, but also aids in the preliminary selection of the various methodologies,

yields meaningful and quantifiable comparison results

is repeatable from project to project,

facilitates the involvement of multiple stakeholders,

aids management in making informed decisions,

considers organisational requirements.

Each of these requirements was reflected upon in order to assess if the process that was presented did indeed meet the requirements.

The well-defined process aided in the preliminary selection of different software methodologies to compare. This was achieved by using the organisational requirements, and matching them to methodologies by considering their underlying principles.

By using these underlying principles, and comparing "apples with apples", the second requirement was also met. The results were also meaningful in the fact that it gave decision makers enough data to make an informed decision.

The results were proved to be quantifiable, and as such, this requirement was also met. Some improvements could be made here, though, in terms of a better scoring scale and weighting calculations. However, the principle still holds.

This process is indeed repeatable, as it takes into consideration the unique nature of the organisation and the industry in which it operates. Other organisations can therefore utilise this process to assess their own methodologies without the need of an external consultant.

The process involved all stakeholders from the beginning of the process until the end. It does therefore meet this requirement as it took into account views form all the different areas in the organisation that had a stake in the end result.

The results were shown to be usable enough to aid in the decision making process. For example, even if the URDAD methodology had a lower score that Aris or RUP, and cost was an important consideration, it still would have been chosen as the differences of the scores were not significant. This shows that more information is available than just the scores, and supports the case for this process to be seen as a decision support system.

The organisation's requirements for a software development methodology were taken into account, by gathering them at the beginning of the process.

The process and framework to compare software development methodologies presented in this paper serves as a *pilot study*. Very few sources are available that address this important activity. The intent of this paper is to stimulate debate and initiate future research in the area of software methodologies. We conjecture that this would considerably increase the possibility of a more complete framework, and processes to apply that framework, to be created by the research community and add significant value to the software industry.

13. ACKNOWLEDGMENT

We would like to thank several members of staff of *Strate* Ltd., South-Africa, for being supportive of the process described in our paper, especially: Eileen Thornhill, Elli Lechtman, Hayley Smith, Janeen van der Walt, Ronel Pieterse, and Mike Higgo.

14. REFERENCES

[1] AVISON, D., and FITZGERALD, B.: *Information Systems Development Methodologies, Techniques and Tools*. New York: McGraw-Hill, 2003.

[2] BERARD, E.V.: *What is a Methodology?* White paper: The Object Agency, 1995.

[3] BROOKS, F.P.: *No Silver Bullet – Essence and Accidents of Software Engineering*. IEEE Computer, 20(4), pp. 10-19, 1987.

[4] FITZGERALD, B.: *An Empirical Investigation into the Adoption of Systems Development Methodologies*. Information & Management 34, pp. 317-328, 1998.

[5] KULAK, D., and GUINEY, E.: *Use Cases – Requirements in Context*, 2nd Ed. Boston: Pearson Education, 2004.

[6] NORTHOVER, M., and NORTHOVER, A., and GRUNER, S., and KOURIE, D.G., and BOAKE, A.: *Agile Software Development – A Contemporary Philosophical Perspective*. SUBMITTED FOR REVIEW: SAICSIT 2007

[7] SCHACH, S.R.: *Object-Oriented & Classical Software Engineering*, 6th Ed. New York: McGraw-Hill, 2004.

[8] SNELTING, G.: *Paul Feyerabend und die Software-Technologie*. Softwaretechnik Trends 17(3), 1997. English translation: *Paul Feyerabend and Software Technology*. Software Tools for Technology Transfer 2(1), pp. 1-5, 1998.

[9] Architecture of Integrated Information Systems. http://www.aris.com/

[10] Object Management Group. http://www.uml.org/

[11] Rational Unified Process. http://www-306.ibm.com/software/rational/

[12] Solms Training Consulting and Development. http://www.solms.co.za/