Critical analysis of MMDIS concepts
Comparing Agile with MMDIS Approaches to Information System Development

Petr Nálevka
University of Economics, Prague
Department of Information and Knowledge Engineering
petr@nalevka.com

Abstract
This paper analyses the Multidimensional Management and Development of Information Systems (MMDIS) methodology contribution to information system development in the context of other commonly used methodologies, especially methodologies which are commonly referred to as Agile.

The aim is to find out what are their similarities and where do those approaches differ especially with respect to software development. The author's aim is further to highlight possible risks and weak points of the MMDIS methodology based on study of articles by prestigious international software engineering authorities and based on authors own over 10 years experiences with real life software projects.

Keywords: Methodologies, Software Development, Agile, Scrum, Extreme Programming, Waterfall, Prototyping, MMDIS

1. Introduction
From the very beginning of Information Technologies as a field, people are facing ongoing failures of the developed software to satisfy original requirements. Projects are getting late, over budge and meet only a small portion of the requirements or in the worse case they are not suitable for the tasks intended at all.

As a result, experts started to impose all sort of very detailed step-by-step development processes to better manage risks of the software projects. Those processes are commonly known as methodologies.

First methodologies in 70s and 80s were inspired by processes taking place in other industries. The production line [AL], a highly effective and manageable manufacturing model, has become inspiration for IT.

Methodologies derived from the production line model break software development into a series of stages where each stage is dependent on the output of the previous one. A typical example is the Waterfall technique first published in 1970 [WF]. The aim is to lower risk by putting every single stage of the project under tight control and define and plan the whole progress in advanced. That's why such methodologies are commonly referred to as plan-driven.

1 Even already the original article by W. Royce claims that the waterfall technique is not intended for large projects, the waterfall approach become very popular in many methodologies which omitted the iterative nature proposed in the original article. This demonstrates that misinterpretation is the biggest enemy of any methodology.

2 Such methodologies include for example Structured Analysis and Design Techniques (SADT), The Yourdon Systems Method (YSM), Specification and Description Language (SDL), Information Engineering and Jackson System Development (JSD), Capability Maturity
Unfortunately, soon, plan-driven approach to software development has shown it's drawbacks when applied in IT. Software projects are in their very nature unpredictable. No fixed plan may be created for them in the first place. Plan-driven methodologies assume requirements for a particular project may be set in stone in advance, but this has shown as not being satisfied in IT. Developed software could not adapt to the ever-changing business reality and in many cases became outdated already at the release date. Moreover, applying a pre-planned process in a field where creativity and ideas are so important made developers become frustrated and less productive.

Object oriented methodologies brought a breath of fresh air into the field in the 90s. They allow better mapping between real world entities and modelled entities in the information system. They do better reflect the fact that the business domain is changing very quickly and the developed software needs constantly adapt to the reality.

OO methodologies were mainly based around UML modelling techniques. See [OOM] for a comprehensive analysis of different OO methodologies.

The object oriented approach allows (if well designed) to encapsulate pieces of functionality in separate components, so that changes in one component do not necessarily impose changes in other components. Also objects allow to define extensibility points for easier change management.

Even the object oriented approach brought more flexibility into the field and maybe even saved the industry from a massive failure, experiences has shown, that it still does not save software projects from missed deadlines and requirements.

It seems as if the problem with object oriented methodologies is they tend to be too focused on the design itself. The main thread is, creating a perfect design may become more important for developers and architects than delivering actual value for the customer.

2. Agile Methodologies

Recent methodologies are heading to overcome some of the above mentioned issues. Such methodologies are commonly called agile to express that they are intended to respond quickly to the ever changing needs and that they are in the very nature designed for flexibility.

Doing agile development means thinking agile. It requires a complete shift in the way we think about software development. Rather than searching for analogies in other industries, agile methodologies propose to approach software development
uniquely as it is really a unique human activity [MF]. They learn their lessons directly from real world successful or failing IT projects.

Although agile methodologies are the latest evolution in how we approach software development, they don't automatically mean better results, see [APM]. Even they had addressed many limitations of their predecessors (the plan-driven methodologies) they do not guarantee success. They do shift a lot of responsibility from rigid rules to the actual people involved in the project. This inevitably means the project outcome highly depends on the quality of the people involved.

No methodology can ever replace highly qualified experts. It can only help the experts not to be burdened too much with unnecessary bureaucracy, rigid processes and purely political decisions [AL].

To quickly summarize the main message of the agile methodologies, we can use the four values of the agile manifesto [AM6].

**Agile Manifesto**

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

### 3. MMDIS

The MMDIS methodology which is maintained at the University of Economics has been introduced in 1992 [MMDIS1]. It differs from the production line methodologies as it does not insist on a strict order of detailed steps. It rather is a proposed way of thinking about the software project. It defines each steps of the project life cycle in several dimensions but rather that explicitly describing what needs to be done in each step, it proposes several topics to think about at each step [MMDIS1].

Since 1992 MMDIS evolved from a methodology which primary focus is on software development to a methodology which tries to cover overall IS/ICT performance management [MMDIS2]. Although its scope is much larger, in this paper the author will focus primarily on aspects related to software development and especially on the traditional custom software development model ignoring some of the latest currents in the field as for example outsourcing, SaaS or parametrized generic purpose software bundles. For more information on those topics, please see [MMDIS2].

Even new software distribution method appeared and become popular recently, software development and its quality is still very important. Whatever distribution approach is used to deliver software to its users whether it is SaaS or any other approach the delivered software had to be developed using some software development methodology. Eligibility of the selected methodology is crucial for the software quality and influences essential parameters as for example flexibility which

---

6 Although those are very high-level principles which may be very easily misinterpreted as stated in [EA]

7 Note that this is true if we compare MMDIS with the plan-driven methodologies, but from the agile point of view MMDIS seems much more strict and rigid.
determines how easy it is to accommodate the developed software to the ever changing business reality.

Even different distribution methods do not lower the importance of software development methodologies, they do shift their target audience. In the era when custom software development was predominant on the market, development methodologies where important also for the larger enterprises who did frequent self-made software customizations in their IT departments [MMDIS2].

New trends as for example outsourcing made companies focus more on their core businesses and thus software development methodologies are more targeted on those companies whose core business is related to software development.

3.1 Dimensions

The main principle of MMDIS as a software development methodology is multidimensionality. Multidimensionality means seeing the IT project from all views relevant to its efficiency. MMDIS distinguishes two main groups of dimensions.

First group of dimensions
- level of abstraction
- the time dimension
- level of IS/IT integration

Integration of
- visions
- outside environment
- business processes
- technology

Second group of dimensions
- information and data (inf)
- processes and functions (pro)
- economic and financial (eco)
- organizational and legislative (org)
- labour, social and ethical (lab)
- software (sw)
- hardware (hw)
- methodical (met)
- document (doc)
- management (mng)

The second level of dimension demonstrates one of the interesting characteristics of MMDIS. Technology (sw, hw) is just one of many other dimensions. MMDIS scope is much larger than only technology, it focuses on all aspects of the entire company to integrate them well with the technology.

This is in contrast to some agile methodologies which focus mainly on the technology and how to encourage IT people to be more productive8.

8 One of such methodologies is Extreme Programming [XP], which on one hand makes customer satisfaction the main aim of the development process and adjusts all the proposed best practices to achieve it, but it is not too much focused on whether the customer requirements are really in sync with their strategic goals which is let to the company to judge.
Also when compared to other methodologies MMDIS sees the software projects from the perspective of the company which actually needs the software — from the customer perspective, rather than from the perspective of an IT consultancy or a software house which is doing the implementation. The second is probably a more common point of view in software methodologies as it corresponds to a common model in the software market where software development is being outsourced [VOR3].

The multidimensional approach shows the IT project is its whole complexity. There are several very different dimensions which all influence the success of a IS/IT project and they even influence each other.

Note that dimensions are not a rigid enumeration. [MMDIS2] views multidimensionality rather as a generic problem solving principle where each dimension is solved first separately and later the separate solutions are integrated into the overall solution.

3.2 The User Perspective

MMDIS emphasises the difference between the user and the implementation perspective. During the implementation process the user perspective gets transformed into the implementation perspective. This transformation helps not only to implement user requirements but it also allows to validate the requirements in terms of feasibility and consistency between all the implementation dimensions.

In terms of software development methodologies, gathering requirements and transforming them into a software project is the crucial and one of the most tricky tasks. To meet the prime objective of the agile methodologies — to satisfy the customer, it is important to understand his needs. This is difficult as in most cases even the customer/user does not know what he needs or he is unable to express it in words understandable to someone who is not heavily involved in the customer's domain.

It seems the process of transforming the user requirements to implementation perspective is in many cases accountable for project failures. Agile methodologies do not give detailed guidance in this case. They rather leave it to interpersonal communication. They emphasise the importance of the adaptive customer [MF] on side and constant feedback on iteratively added features.

MMDIS on the other hand bets on a precise project plan which is validated against the high-level strategic goals of the company, the overall IT concept, which has to be consistent across all the dimensions and in all stages of the project life-cycle and which is shaped by the company management, partners, customers as well as by individual users9. Before implementation takes place, the whole complex space of requirements is transformed into the feasibility study document and into detailed conceptual design.

3.2.1 Management

One of the most important user perspectives in MMDIS is the management perspective. MMDIS is explicit about the role of management in terms of

---

9 This is a nice goal, but it may be very difficult to achieve. It seems as MMDIS is not very verbose on how to achieve such kalokagathic state of the project.
constraining time and financial budget and defining the target business processes which shall be supported by ICT. As nobody else is eligible to do this kind of decisions, management user perspective has the highest priority and it is very important to clearly understand aims of the management.

An important aspect is involvement of management in the whole implementation process. Without management involvement the project would soon end lacking decisive power.

Agile methodologies especially highlight the fact that gathering the requirements from the management user perspective is not a one time activity. This is something which needs to be evaluated constantly and management needs to be constantly confronted with the current progress of the system[10]. If management is not able to make the needed decisions on-time, delegation of power is highly required. It is especially important to have an eligible customer representative on side with a sufficient level of decisive power [XP] [SCRUM].

One of the models for transforming customer requirements is the quite common fixed price contract where requirements are specified in the beginning of the project and price bids are accepted [MF]. This model seems very much vulnerable to failure but it is also the model which seems as a natural choice for MMDIS, as it emphasises the role of the feasibility study and gives the management the obligation to set not only requirements, but also to allocate resources and financing.

This model may be sufficient for the production line industries, but not for software development which (as we already stated) has a highly unpredictable process[11].

In today turbulent markets it is very likely that the requirements change significantly before the end of the project. Also some requirements become obvious not sooner than during the project implementation. That's why the agile approach rather recommends an iterative approach where in each short-time stages management may evaluate the progress and change direction of further development (including changes in time and resources allocation). This is in contrast to fixed price contract and it requires both an adaptive customer [MF] as well as an adaptive IT supplier.

Although requirements are high, there is a chance that the project will end in a satisfactory manner for both parties, in contrast to the fixed price contract where both parties usually end as losers [MF].

The MMDIS methodology emphasises how important it is to understand the management aims, but it is not very verbose on how to do that. Also it does not stress what agile methodologies are stressing very intensively, that understanding the aims in an on going process which involves constant reevaluation during the whole project progress.

3.2.2 The end-user

In MMDIS the end-user is a source of valuable very detailed (in comparison to the management user perspective) information. As they expect the IS to support their

---

10 Ideally they shall be confronted with a running prototype application.
11 Some methodologies work with a predictable process (especially the plan-driven methodologies). As predictability is a very desirable feature for management it is always very tempting to use a predictable methodology. But using a predictable methodology on an unpredictable field is very likely to cause project failures.
Every day activities, the implementation experts are encouraged to gather all sort of important knowledge from the end-users.

**End-user knowledge [MMDIS1]**
- processes on the particular user level
- who is responsible for a particular stage in the process or for the process as a whole
- what activities are candidates to automation
- what data needs to be collected and persisted

The agile methodologies add one more role (but a crucial one) to the end-user. The end user is a valuable source of evaluation whether the current prototype is of any customer value to the particular users. The end-user is seen as a first class tester of the system and implemented features are consulted on a regular bases with the involved users.

### 3.3 Implementation dimensions

The section called “MMDIS” describes two different groups of MMDIS dimensions which are relevant from the implementation point of view. The first group of dimensions defines characteristic phases of implementation.

**By separating implementation into phases MMDIS aims to [MMDIS1]**
- separate work into projects for better planning and management
- allow some form of incremental development or prototyping
- divide tasks based on the level of abstraction to be solved by different experts
- ensure compliance with the high level strategic aims of the enterprise

**The main phases [MMDIS1]**
- Feasibility study
- Global analysis
- Detailed analysis
- Implementation
- Installation
- Production and maintenance

### 3.3.1 Incremental development

The most important statement from the agile point of view is the MMDIS’s emphasise on prototyping and incremental development. But in the same time MMDIS introduces very detailed and sequential phasing of the project’s life-cycle. Phases may be misleading as people may immediately thing about phases as sequentially executed steps where the next step may be executed only after satisfying all the prerequisites of the previous step.

Because another significant source of IT project failures is misinterpretation of methodologies [AL], it is important that methodologies state clearly their intentions to forgo such misinterpretations. This is a double-edged issue, as making the methodologies too explanatory makes them on the other hand too rigid and less applicable on different projects.

---

12 According to MMDIS, all of the phases need to be compliant with the global enterprise strategy and IT strategy
Because of the danger of misinterpreting phases as being something fixed, agile approaches usually tend to either recognize only the very high level phases or explicitly and clearly explain that phases are in their very nature highly parallel, iterative with huge overlaps and executed in a very short time periods (rather days or weeks rather than months and years).

From this perspective MMDIS reminds some of the plan-driven methodologies which define a rigid waterfall-like process but as of the influence of agile they try to extend the same rigid process with agile elements like prototyping and iterations just to keep their methodology trendy.

The problem with this approach is, the iterative and parallel nature of the development processes has to be embedded into the very essence of the methodology. The whole methodology should be build up around this idea, otherwise there is a risk the individual phases will be executed in a long run sequential manner and thus making the project miss the requirements which will change or become more clear in the mean time.

Modern methodology needs to have mechanism which will constantly challenge the achieved, constantly reassure the requirements and constantly deliver software increments in the shortest possible integrations.

In the section called “Introduction” we described three types of methodologies, the plan-driven (manufacturing, production line), object oriented (oriented on design) and agile (customer-oriented). In [AL] different methodology types are mapped to people’s three fundamentally different ways of thinking about software development. Having well described sequential stages is very appealing especially to people of the manufacturing group as it creates an illusion of software development as an predictable process which it is not at all.

In [MF] Martin Fowler says: “This leads to people using a methodology in the wrong circumstances, such as using a predictable methodology in a unpredictable situation.” This is exactly what happened with the plan-driven methodologies of 70’s and 80’s. Allowing people to process the methodology in a rigid way (and not shielding them from this seductive option) will inevitably increase the project risks as predictability in software development is only an illusion.

### 3.3.2 Implementation roles

Even more dangerous is to explicitly bind the project stages with a certain group of experts as suggested in MMDIS. [MMDIS1] suggests to “Clearly divide individual levels of abstraction when implementing an IS/IT solution... and thus determine position for individual experts during implementation...”. This is very dangerous as it may be understood as dividing people strictly into managers, architects, analysts, developers and testers. In the worst scenario, those groups are disjunct and hierarchical.

This idea is again very appealing especially to people with the manufacturing and partially also design mentality. The manufacturing mentality would like to see involved people as replaceable units (see [MF] and [AC]) as only with such people...
Critical analysis of MMDIS concepts

the project plan is predictable. In their eyes projects dependent on individuals are
highly vulnerable to risk.

This is in a direct contrast with what agile methodologies claim. One of the four
main statements in the agile manifesto [AM] promotes “Individuals and interactions
over processes and tools”. In general many of the agile methodologies are strictly
people-centric. The idea is, if managers treat people as replaceable units, they also
end up with replaceable units [MF] and this will inevitably influence the quality of
the results.

Replaceable units are of very little use in a field such as software development
which is highly creative and dependent on the intellectual capacity of involved
people. Such approach would lower morale and commitment of people. This
approach resembles Taylor's scientific management and it may be useful in a
factory at the production line, but it can hardly bring any good for software
development.

Even substitutability of people may help managers to feel safer, it also inevitably
lowers productivity to a fraction. A company practicing such approach may hardly
survive in an highly competitive market segment as IT.

Rather then anything replaceable, software developers should be treated more as
artist or scientists [AL]. They usually are highly knowledgeable professionals with a
high level of self discipline. The role of the managers is not to control those people
but to help them get the required competences and to overcome political and
organisational barriers which hider them from being able to execute.

In contrast to creating replaceable workers, Agile methodologies propose to hire
the best professionals and give them competences and the power so that they can
use all their potential.

Making time and resource planning by the managers, conceptual design by the
architects, detailed analysis by the analysts, code writing by the developers and
testing by testers is the shortest way to a very inefficient process.

Managers will never have the necessary technical information to do any time
estimates. Architects can hardly create a perfect design without feedback about the
underlaying implementation details. Analysts will never realize all the low level
implementation difficulties without touching the code physically. Developers will be
frustrated with the design which forces them to create ridiculous code, to implement
structures which looks nice in the high level concept but do not have any meaning
and only complicate things on the implementation level. Testers will create tests
which pass and which are easy to maintain rather than test capable of finding
errors. They will never test all the functionality intended by the developer because it
is not their code they test, they don't see all the nuances and all the intentions
behind.

The morale of the above example is — ideally project management, design,
analysis, development and also testing shall be done by the same people. This is of
course highly resource demanding and it can hardly be imposed in the real world.
But at least the aim is to make larger overlaps within those categories so that
designers are also involved in project management and they also do at least some
programming and they are familiar with the most important peaces of code.
Developers are heavily involved not only in analysis but also in design and they
write tests for their own core.
This approach also leads to use of rather smaller teams as IT professionals are a highly non-scalable resource [AL] [MF].

Some may argue that overlaps and small teams may hardly be achievable for large projects where too many people are involved and the overall picture of the project can hardly be kept in minds of all of them. On the other hand we need to clarify what is actually a large project and how we recognize one. Definitely we can’t use a circular definition and claim that large projects are those where a lot of people are involved. The author of this article has witnessed and participated in few “large projects” which had been implemented in very small teams composed of highly knowledgeable self-disciplined professionals. There is an immense efficiency gain in this model when compared to producing the same result with replaceable over-specialized workers. Achieving same result with them would be either highly inefficient or even impossible.

In some cases software projects really need involvement of more people to get them done in a reasonable schedule. Then it is really an issue how we approach scaling of resources. Rather than structuring people vertically into subordinate roles it may be more efficient to divide big projects into small partially independent sub-projects driven by small highly capable teams with responsibility overlaps and execute them in parallel. Consistency of the overall project is then maintained by intensive communication of the teams responsible for the components who have mutual dependencies.

With small agile teams managers may feel not having control over the project risk management. That’s probably one of the reasons why they seek predictability through a rigid process and replaceable human resources even they have to pay for it with immense productivity drop.

There may be a more efficient way of controlling risk using agile teams. For mission critical projects two or even more competitive teams may be established to achieve the same goal. Results of all teams are evaluated and finally the better result is used in production. This approach makes people highly motivated and it may also serve as a way of evaluating teams productivity.

Still there are many companies on the market capable to survive with the inefficient replaceable unit model. It is difficult to find reasons for that but let’s just mention few possible causes:

- The lack of knowledgeable self-disciplined professionals.
- The distortion on the market which is very strong in case of IT. The customer is not capable to accurately evaluate the quality of purchased software especially in short term.
- The replaceable unit model may be slightly more predictable which could be crucial for some business models. Also it may be appealing to customer with the manufacturing mentality.

At this spot it is worth mentioning offshoring as a popular method to lower costs of software development. From what has been stated above it is questionable if offshoring may bring the same benefits it brings for other fields also in IT. Again we

---

14 Large in terms of functionality covered and internal complexity of the system. Those projects basically involve implementation of very large systems similar in scale to what has been developed by hundreds or even thousands of people for several years by the competition.
observe the IT field is really unique and very different from other models. What works well in manufacturing may fail terribly in IT. Even cost of human resources in some countries may be a fraction of the costs in the developed world, the success of offshoring very much depends on the human capital you are getting for that cost; on their knowledge, creativity and attitude to work.

3.3.3 Levels of abstraction

The MMDIS methodology further proposes to clearly divide individual levels of abstractions of the IS/IT to make it better understandable [MMDIS1]. This may sound nice, but it also may be quite risky. We need to keep in mind that the more abstract constructs we define the more blurred picture of reality we get.

Over the last decades Business informatics has establish a whole new language which helps non-technical people in IT to communicate with non-technical people in other areas in order to sell IT services and software licenses to them. This language composes of trendy words¹⁵ and diagrams describing the IT/IS at the highest level imaginable (the highest level of abstraction). Such diagrams have nothing to do with what is the IS/IT really composed of and how do the individual components interact with each other.

The purposes of the business IT language is to evoke in non-technical people who make IT purchase decisions that they have some understanding of their IT system internals (but this is of course a pure fiction). If managers get an delusive feeling they understand the IS architecture they may start managing it; moving boxes/components all around and filling missing gaps in the diagrams by buying new components and “enhancing” their current IS/IT architecture.

What they are doing is not enhancing their company information system, but they are enhancing the appearance of the high level diagrams which are so far from the real information system, that they have nothing to do with it any more. As a by-product they are paying huge money to IT businesses.

Figure 1, “Building block of information systems” shows one of the semantically empty diagrams commonly used in Business IT. The level of abstraction is so high the diagram contains no information about the actual architecture of the IS/IT. It is rather a psychological tool which shall address the fears and desires [AL] of the management deciding about purchases of new IS/IT components. It displays the information system as a hierarchical entity which reflects the organisational hierarchy in the company which the management desires to preserve. The IS/IT may be viewed as a tool to preserve their superiority as it provides specific components which are aimed only for their hierarchical level and the other components intended for the lower management or non-executive staff are somehow subordinate to the monumental EIS (Executive Information System) top of the pyramid.

¹⁵ Those are commonly referred to as buzzwords. The set of terms to describe the same thing in IT usually completely changes over a year so that same things are described with completely different words but those words are trendy and appealing. The aim is to achieve an impression of innovation and high tech even the only novelty came from the marketing teams.
Also the IS/IT is viewed as composed of many different building blocks magically related to the different activities and departments present in the company organisation structure. This again addresses the management fears and desires. The fears that the construction is imperfect and some building blocks are missing which may disrupt all the magnificent building. Or the desire to have a perfect pyramid with all blocks nicely placed at their intended position, to have a better and more stable pyramid than the competitors.

The real quality of the IT/IS, how it is integrated with the business processes, how it is useful in everyday tasks of the company departments is hidden in a tiny sarcophagus deep inside the pyramid where no manager can see.

There are more Business IT examples of diagrams which are misleading or even creating a false view of the IS reality which may be highly unhealthy if we forget about how abstract is the level of our real understanding. Loosing touch with the reality because of being in a too abstract level is not related only to non-technical people. The very same issues may be experienced with architects which have lost contact with developers and the code. They are living in an artificial world of their UML or other design diagrams. They suffer a delusion that any entity they draw in

---

16 This diagram was taken from [MMDIS1]
17 Another example are the famous Garthner magic quadrants. From the authors personal experience the more is a certain product judged as being visionary and the more as being able to execute the more ugly is the technical solution of the product, the more non-standard components are used, the more time will it require to integrate the solution into the company's IT infrastructure and of course the more money will be spent on the licenses and the more will cost the consultations.
their diagram gets automagically implemented in the very same scale as they have intended.

They are constantly surprised that some aspect of the system diverges from their design and blame the developers for that. On the other hand, developers feel the architects are completely out of picture and their contributions bring more obstacles than help to the project progress.

3.3.4 The project life-cycle

MMDIS describes the project life cycle as repetitive iteration of the main project phases, see The main phases [MMDIS1]. It does not explicitly say that the phases are sequential but it is also silence about how parallel and interwoven those phases are.

Feasibility Study

From the agile point of view it is ok to recognize difference in the activities during project implementation but it needs to be clarified that those phases are highly parallel and they constantly influence each other and their outputs need constant reevaluation. Otherwise there is a risk that manufacturing mentality people [AL] will tent to set requirements in stone during feasibility study, or that design mentality architect will tent to capture persistent design schemes during global analysis.

Although it is not explicitly stated, the whole MMDIS project life-cycle approach feels like a plan before you build approach (see [MF]). For example, even MMDIS proposes an iterative approach in case the project tents to be too large, the incremental subprojects shall be planned in advanced in the feasibility study. Plan before build is a good approach in mechanical engineering but according to years of experiences it is completely inappropriate [AL] [MF] for IT. One of the issues is the IT project design may actually look very good on paper but because of the issues with abstraction partially described in the section called “Levels of abstraction” the real implementation issues and drawbacks cannot be predicted in the design stage and they will most probably emerge afterward not earlier than during implementation.

Also in engineering areas (as e.g. bridge building) , the design phase is only a fraction of the overall time, resources and cost of the whole project. But according to [MC], in IT the cost of design is highly over 50%. That's why design and implementation (actual coding) in IT is a parallel interwoven process which can hardly be separated. Of course those processes can be forced to be separate, but the only achievement will be a more time and money consuming project with loads of design documentation as a by-product. Such documentation will anyway be — in the better scenario — ignored and — in the worse scenario — forced to be followed.

In their seek for predictability, companies often use the fixed price contract business model with fixed requirements agreed on in advance as mentioned before in the section called “The User Perspective”. On one hand it gives the illusion of predictability but it is one of the most common reasons for IT failures. Usually the victims are on both sides. The development company losses the contractual money for not satisfying the requirements and the business company looses time and maybe even market position as their core business processes are not adequately supported with the IT system as they require it [MF].
Fixed price contract is a proper solution in building industry or workmanship\textsuperscript{18}, but it is inappropriate in IT which has completely different characteristics.

The agile answer to this problem is you simply can’t fix all properties (time, cost and scope) of the project in advanced [SA]. Usually you fix the budget and time and keep the scope flexible (but controlled).

![Triangle Diagram]

**Figure 2. The iron(y) triangle [SA]**

This approach is accompanied by incremental development with a heavily involved customer providing constant feedback. Such model has many advantages for the customer. Especially very early production results and constant feedback about the project state. On the other hand, this model is not compatible with fixed planning in advanced. We would experience constant diverging from the fixed plan, similar to what will be experienced in the fixed price contract scenario but with fatal consequences.

In contrast, this model is constantly adapting to the ever changing project state. This means even we can spot stages as feasibility study or global analysis highlighting them as phases may be misleading as those are constantly questioned and reworked during the whole project progress.

**Global and detailed analysis**

MMDIS explicitly separates the phase of conceptual design analysis and detailed analysis. This division may seem artificial and there is very little advocation for it in [MMDIS1]. It seems like this division has been made only because traditional software development methodologies (especially the object-oriented methodologies, see the section called “Introduction”) do it also.

As stated above the section called “The project life-cycle”, design is more than 50\% of the project cost. We have to decide if we really can afford the luxury of a separate conceptual design phase. Also as we already state, design is not a one-time activity, it needs to be updated constantly during the project life cycle. This means our conceptual design needs to be constantly maintained and kept synchronized with the ever changing requirements.

\textsuperscript{18} It is in interesting coincidence that [MMDIS1] really demonstrates the project life-cycle on a building industry metaphor.
The question we have to ask at this stage

- What extra resources will it cost?
- What benefits it brings?
- Does conceptual design bring more quality to the project?
- Are the benefits outweighing the costs?

According to the authors' experience, conceptual design is usually an overhead even in large-scale projects. After a while when deadlines get tight, conceptual design brings only frustration as everybody in the team can clearly see how far did it diverge from the current state of the project. As such, it can hardly be used as a reliable source of documentation. It becomes purely a burden.

Although not very useful for the project itself — to be fair — it needs to be stated that there are political reasons why conceptual design may be useful. Especially to proof the customer we use a well-planned development methodology which may be very much appealing to customers with a manufacturing and design mentality.

In contrast to MMDIS, Agile approaches are usually quite strict about discouraging people from doing too much documentation. For example in XP, a complete conceptual design is exactly the kind of think you should avoid. Although there may be some exceptions, for example a quick design meeting, but in this case you should:

- keep them short
- don't try to address all details
- treat the resulting design as a sketch, not as a final design

Martin Fowler claims: "When you do some up-front design, you'll inevitably find that some aspects of the design are wrong, and you only discover this when coding. That's not a problem providing that you then change the design. The trouble comes when people think the design is done, and then don't take the knowledge they gained through the coding and run it back into the design".

The author of this article has witnessed several projects where tons of documentation have been produced by IT constancies in years without actually producing any line of code. Such a L'art pour l'art approach to design is a pure disaster for the customer who may be spending a fortune for nothing useful.

Traditional plan-driven methodologies (see the section called "Introduction"), with a similar project life-cycle to MMDIS, are to be blamed. The ever-changing requirements do not allow teams to ever leave the conceptual design stage. Designing consumes all their time. They keep adjusting the design so it is never finished to be handed over to implementation and in a plan-driven methodology developers can't really start their job unless the design is perfectly polished.

**Implementation**

In contrast to the agile’s detailed focus on implementation techniques, MMDIS is not very verbose about this phase.

---

19 They automatically assume that if the plan-driven processes work fine in their industry they are in a similar manner applicable in IT.

20 Although there are some examples of a co-existence of conceptual design with agile. Also [MF] argues that documentation is not automatically the source of all evil.
It seems as if MMDIS would anticipate all the difficulties and issues are concentrated in the preceding stages but implementation is only a workmanship activity which mechanically translates the all-explanatory design into code.

The author of this article has seen brilliant designs turned into junk by non-professional developers. Even with a perfect design (and this is really an utopia, see the section called “Feasibility Study”) still, most of the technical problems need to be solved during implementation. If they are solved in an unsatisfactory manner the perfect design as well as the whole project may very easily get ruined.

That’s why developer’s quality, their creativity, commitment, team spirit and knowledge is absolutely crucial to the IT project’s success. In the real world, implementation is very often perceived similarly as presented in MMDIS as a mechanical activity. Those are the roots of a huge underestimation of the role of developers and importance of code quality.

This leads to a very unhealthy situation in the IT industry where coding is seen as an inexperienced activity which is proceeded by students, brigadiers or junior people in general. Everybody tries to skip this disdainful sage of their career as quickly as possible and they are heading to become more respected as analysis, architects or project managers.

The most important product of the software development becomes the most marginalized element of the project. As a consequence, the quality of software in general is declining21.

3.3.5 More dimensions

There is a second group of dimensions which according to MMDIS has to be taken to consideration in every stage of the project life-cycle, see Second group of dimensions. At this stage MMDIS is basically enumerating all sort of essentially disparate knowledge which shall not be omitted to achieve an successful project and which is glued together into entities called dimensions.

MMDIS aim is probably to demonstrate how complex an IT project domain really is and what all different aspects we need to take in consideration. This may have a Wow-I-did-not-think-of-that effect on IT beginners, but on the other hand, most of the highlighted points are common knowledge to IT experts or they are very high-level to have major impact on the way IT projects are managed.

For example in the data and information dimension MMDIS highlights the notorious definition of data, information and knowledge, although it provides some useful

---

21 This is the author's personal opinion, there can hardly be any proof for that as software quality is highly intangible. But from the authors experience many companies started as geek’s undertaking which a perfect product with a very high ratio of technical people. But as a result of a typical life-cycle of such start-ups, the ratio of technical people starts to decrease in favour of management, marketing and sales, who are taking more and more superior role in the business.

As a result the product quality decreases, innovation stops and the only thing which persists are the business people who try to exploit the original idea to the limit, but they are of course unable to bring any new ideas in.

Fortunately for the industry there is usually already another bunch of geeks grounding another brand new start-up. If they are lucky enough the business spongers won’t occur too early. Let’s keep fingers crossed for them.
insight on different information characteristics which shall be kept in mind and on the cost of keeping a certain information in a the system.

As part of the process and function dimension MMDIS proposes to define hierarchical decomposition of functions of the IT/IS from the high-level functions to elementary functions and to define inputs, outputs and control data for each function. The same is true for processes which may be modelled in a hierarchical manner.

As mentioned in the previous sections, such detailed documentation may be a big overhead for most projects and it may be very costly to keep such documentation in sync with the changing reality. That's why Agile methodologies usually propose much more lightweight approaches to capture customer requirements.

For example Scrum works with the product backlog which contains detailed description of the high-level required features and wish lists which are prioritized by business value. Such documentation is much easier to synchronize with the ever changing requirements than heavy-weight UML diagrams which needs to be synchronized first before changes may be propagated into the code.

A very significant portion of Agile methodologies focus on methods to capture user requirements in the code directly and how to manage constant requirement changes there. The relevant techniques are: refactoring, unit and functional testing, keeping the design clean when refactoring rather than hacking new features in etc...

Another reason for detailed documentation is the risk of losing important people on the project and the need to train new stuff. Again in this case functional and process diagrams would be of any use only if they are perfectly in sync with the reality. Much more useful in this situation is verbal guidance from the team members, customer requirements and of course the code is the ultimate documentation for the professionals.

The management dimension assigns role, resources, responsibilities and does the overall planning. It defines participation of IT specialists, top management and end-users in the project.

In general MMDIS defines the aims of the management dimension as to optimize resource (people, financial...) consumption to achieve the planned IT/IS requirements on time.

---

22 MMDIS actually does not give a project scope and scale when it is relevant to do such detailed documentation. This may lead to an impression that it is proposed in any scope and scale which would be ridiculous.

23 The backlog is constantly updated and synchronized with the customer by the product owner - one of the Scrum roles which represent the interest of the stakeholders, usually working on the customer side. The development team on the other hand assigns estimated development effort to the backlog. The features with the highest business value first and the shortest effort estimate are implemented first. The customer requirements - called also user stories (small pieces of papers containing 1-2 sentences of captured requirements edited by customer representatives) are directly transformed into code and both serve as the project’s ultimate documentation.
The Agile methodologies are much more verbose on the role of management in IT projects. The relationship between management and developers is very fragile and crucial to the project success. As a result of many examples of hostile management [AL], Agile often emphasises that management shall not interfere into matters which are only up to the IT professionals such as choice of technology (hardly any manager may understand the technology better than IT professionals) or time estimates to develop a certain feature [MF].

On the other hand, even managers shall not interfere into the matters of the development team who shall not play a passive role. Only managers have the authority and power to help the team in matters where their power is insufficient. [AL] says: "Here, the manager uses their authority and political clout to ensure that those outside the team play fair, to give the team every chance of succeeding. For example, tackling bureaucracy, boosting insufficient budgets, and repelling stubborn competitors and saboteurs. That is, rather that directing their authority inwards to command and control the team, the manager-as-referee directs their power outwards to support the team, removing obstacles that the team is powerless to address themselves."

For example, most of the Agile is based on communication and development teams need constant access to business domain experts for consultation. The role of management is to arrange such communication channels [MF].

Another important role of the management is leadership. The role of the team leader is helping the team find shared beliefs around which to gel, and helping the team succeed repeatedly in their work [AL].

Further dimensions described in MMDIS highlight some facts about IT systems which are obvious to people who participated in few IT projects. For example:

For labour, social and ethical dimensions MMDIS emphasizes the importance of training. Even a superb IT/IS has no value for the business if there are no employees able to operate the system [MMDIS1].

There needs to be sufficient hardware to run the IT/IS with an appropriate number of users [MMDIS1].

The software dimension teaches us that software is composed of software components. The choice has to be made whether to use standard components which are already there or to develop them from scratch [25].

The economic dimension is really crucial as it asks the question whether the gains of the IS/IT really dwarf the costs.

The methodical and documentation dimension defines tools and techniques which shall be used in the project and the kinds of documentation which is produced during implementation and maintenance.

One of the methods used during software development is in fact the methodology itself. [MF] emphasises the importance of accepting a methodology by the team.

---

24 The choice of technology can easily become a political decision. This leads to a huge risk of project failure. Only the IT professionals are capable of deciding whether the requirements can or cannot be fulfilled through a certain technology [MF].

25 An important question is also whether to use open source components which are usually more transparent, customizable and maintainable or whether to use third party components and rely on the third party support and loose control over the particular component.
Critical analysis of MMDIS concepts

rather then imposing it by the management. Accepting a methodology needs commitment but this can hardly be achieved through imposing anything. This will only lead to a stage where the methodology will be violated whenever possible.26

3.3.6 Dimensions summary

According to MMDIS various dimensions in various stages of the project life-cycle do have stronger or weaker dependencies on each other and their importance in a particular stage varies.

To achieve a consistent and integrated IT/IS, no dimension may be ignored, all the dimensions need to be taken in account in all the project life-cycle phases [MMDIS1].

On the other hand this may be very difficult to achieve and all the multidimensional approach stands and falls with the fact whether the idea of propagating the high level strategic goals through all the life-cycle phases, through out all the dimensions is really something which is achievable in a real life project. There are so many levels, so many abstractions on the way that one has to ask whether the strategic visions don't get somehow distorted on the way down to implementation.

4. Main principles

[MMDIS2] further generalizes some of the MMDIS concepts mentioned in previous sections into 11 main principles of information system management.

Principles
- Multidimensionality
- Integration
- Layering
- Flexibility
- Openness
- Standardization
- Cooperation
- Process Management
- Learning and growth
- Resource localization
- Measurability

Principles as a new concept introduced in [MMDIS2], seem to the author of this text as an aim to generalize different MMDIS aspects into high level commonly applicable approaches. This makes MMDIS less controversial and more flexible. As stated in [AL] a common issue in methodologies is to find the proper level of specificity.

26 [AL] has a similar opinion. We already mentioned his division of people mentalities into manufacturing, design and customer cultures. Further more he states — it is difficult to use an agile methodology in a camp with dominant manufacturing or design methodology as well it would cause incredible issues to apply a waterfall process in the customer culture camp. The reason is, people won’t be committed to a process which is in contrast to their worldview. Changes of the culture in a company is possible but very painful and risky.
A methodology being too specific can never cope with the complexity of the real word. All possibilities can never be enumerated and the methodology can never be made suitable for all sort of different applications. Methodology maintainers trying to predict any possible situation and provide guidance are creating methodology monsters which are supposed to be suitable for all sort of very different tasks but failing to do any of them in satisfactory manner.

On the other hand being too generic means the methodology can provide hardly any guidance in any specific situations. Telling only generally accepted facts can hardly be questioned, but such methodology isn’t very useful either.

Some of the statements in [MMDIS2] explaining the principles tend to be quite high level and thus showing characteristics of being pretty obvious or even tautologic in some situation.

For example in order to succeed in creation of an information system the multidimensionality principle tells you not to omit any of the dimensions critical for succeeding.

Integration principle basically says that a system is not just the sum of its parts. Layering is a common principle in IT. It is hard to imaging any IS without layering applied so it is really questionable if this is worth highlighting as a main principle of any methodology.

On the other hand the flexibility principle mentions an very important and controversial issue in software development. Beside the two possibilities mentioned in [MMDIS2] there are other options to gain flexibility.

Some Agile methodologies as XP for example propose to keep code specific and only implement the required features (features which have a relevant customer value) at a time, but keep the code clean so that refactoring and changes are possible in the future without being too painful.

Object oriented methodologies (as mentioned in the section called “Introduction”) on the other hand propose to define extensibility points designed by architects featuring prophetic abilities. But this approach did not show itself as being effective as prophetic architects are quite scare on the IT labour market.

Making the IS flexible through parameters (as proposed in [MMDIS2]) is definitely an option but extremely expensive. Also the aim of parameterizing every aspect of the IS conducted one of the most unhealthy phenomenon in IT history27 parametrized software monsters such as products made by SAP, Siebel, Portal, Oracle and others. According to author’s experience such software products are common cause of software project failures28.

Parameters are great at presentations. They create an illusion of flexibility. With few mouse clicks it is possible to completely change the behaviour of the system to better suit the customer’s business needs. This usually does impress non-technical people who start to believe that changing parameters of their company ERP is like

---

27 This is author’s personal opinion, based on his IT industry experiences.

28 The author’s claim is among others based on his own experiences with a tiny Sibel project which did at least 10 times exceeded it’s original budget and time-frame and after 3 years of implementation it is still under development. A similar project could have been implemented from scratch in few week time.
configuring their Outlook. They see more control over the software and less cost on IT experts, but they are terribly wrong.

The opposite is true. Control over the software shrinks only to aspects which are configurable through parameters and changing those parameters may have unpredictable results as the functionality influenced by the parameters is a black-box\textsuperscript{29} or the customer who now solely depends on the vendor's documentation provided.

The quality of documentation is usually very bad as there is no incentive for the software vendors to reveal any details which could help third parties to configure their software. On the contrary, they wish only themselves or their certified partners are able to customize the software which is lucrative service.

Where source code is written in a standard language and thus understandable to a wide range of IT experts, parameters are a proprietary knowledge and experts need to pay to the software vendor to gain such knowledge through certifications and trainings.

This helps to creates an well thought encapsulated ecosystem where the customer gets enslaved in after he makes the purchase decision. Every time he needs to accommodate the IS to his changing business needs he pay for it either to the software vendor himself or to somebody appointed by the vendor who did already pay for this privileged and now he wants to get his investment back.

The whole system is supported by big consultancy companies (e. g. Garner) who recommend such software vendors as market leaders. At the same time their recommendation serves as advocacy for bad purchase decisions on the customer side. No matter the customer replaced their previous software solution which he had full control over with something which did not bring any additional business value, no matter it is now twice as costly to maintain the software and expensive external consultants need to be paid to do so, but we did choose a market leader so our decision can't be questioned!

Parametrized software is not only good business for all involved parties except the customer, it is also an ugly technical solution. Object oriented paradigm gave us the tool to model our domain directly using objects so that our solution is understandable, clean and flexible to changes. With parametrized software this principle gets distorted. We do not create a domain model directly but rather a generic meta model which is intended to interpret the actual domain model expressed in a proprietary language. This introduces additional complexity, makes implementation much more difficult and the overall solution less transparent just to chain the customer to the vendor.

To conclude flexibility can hardly be achieved through a parametrized software but rather through a good design and availability of the source code and this is related to another MMDIS principle - openness.

With respect to openness [MMDIS2] does not mention the important role of open source software. Having full source code of your IS component gives the company flexibility and freedom of choice. In contrary using a closed source solution may bring better software quality in some cases, but as mentioned earlier it means high risks of being too dependent on the vendor.

\textsuperscript{29} Because the provided software is closed source.
On standardization [MMDIS2] claims: “a software solution gets cheaper and less complex when standards are used”; but this statement can hardly be generalized. It definitely depends on the problem and mainly on the standards involved and their quality.

For further information on MMDIS principles, see [MMDIS2].

5. From development to management

At the end of 90’s MMDIS transformed from a methodology which primarily focuses on development of information systems to a methodology which has a larger scope [MMDIS2]. Development is seen as only a fraction of the overall effort to manage an information system and IT services within companies.

On the other hand it is important to emphasis that the quality of the developed software highly depends on the development process. Moreover the software's quality highly influences the quality of the overall information system and how it is capable to support the company processes. Using the MMDIS language, software development process is one of the most important dimensions which can't be omitted when taking about information systems. Marginalizing software development and it's importance may have similar effects as underestimating the implementation phase as discussed in the section called "Implementation".

[MMDIS2] did not bring any significant updates to the software development aspects of the methodology depicted in [MMDIS1]. It only generalizes some of the approaches as for example the high level principles of the methodology (see the section called "Main principles"). This means a company willing to practice MMDIS who is doing also software development (not outsourcing it) needs to adhere to principles which haven't been significantly updated for years and which ignores most of the recent development which has especially been set on in the Agile camp.

Anyway the management extension to MMDIS it out of scope of this article which primarily focuses on software development methodologies in the narrow sense. For more information, see [MMDIS2].

6. Conclusion

The uniqueness of using MMDIS for software development originates from structuring aspects of IT projects into dimensions and focusing on aspects which according to [MMDIS1] has been neglected in other software methodologies such as the organizational, labour and economic dimension. But the question here is: Is this really a sufficient value added to form and maintain yet another methodology?

MMDIS is ignoring many aspects which has been researched in the Agile camp over the last two decade. Although agile inventions such as prototyping or incremental development are mentioned within the scope of MMDIS, they are mentioned independently on the rest of the methodology only as an extension, not an integral part of the project life-cycle.

Even MMDIS is not a typical plan-driven methodology there is a risk of potential practitioners to understand it as such. On the other hand the plan-driven essence gives it an interesting marketing bonus as it may sell well to people with the manufacturing and design worldview. Even MMDIS could be possibly executed in
an Agile manner, there is very little in the methodology itself which would encourage users to do so.

Ignoring the above, MMDIS is still an interesting reading. It is a nice compilation of advice and precepts which obviously have their roots in practical IT experiences. Also MMDIS holistic approach to IS/IT is quite rare and interesting. Where other methodologies focus on the development process or on the software itself, MMDIS tries to see the Information System as a whole in all its complexity (dimensions) within the organisation structure of the company.

It is interesting to state that the Agile and MMDIS share one of the main principles. The holy grail of Agile is customer satisfaction which is embodies in MMDIS through the ubiquitarian effort the keep the developed system at every stage and in every detail conforming to the high-level strategic vision of the company.

MMDIS is trying hard to explain how high level strategic visions along with other customer requirements get transformed from one project stage to another, from abstract constructs to the detailed ones through all dimensions, so that finally every elementary function in the system is in sync with the global strategic vision of the top management.

This sounds really amazing, but also it is difficult to believe that something similar is possible in everyday IT projects. On the other hand again it is great marketing. What may sound better to the ears of managers than a methodology which is somehow able to encode their beautiful visions in the very genetic essence of their company Information System.

To conclude, even MMDIS brings some interesting ideas the author of this article would recommend it as additional knowledge to be read but not to be used in most of the every day software development projects (even this depends on the scale and scope of the projects and on how fixed are the requirements). On the other hand MMDIS is a perfect marketing methodology. It is common routine for IT companies to have two or more methodologies. One which they really practice and other methodologies which they use as a marketing tool for certain type of customer, because they know they will appeal to their type of thinking or they are required by the customer.

Practicing Agile helps the company to be competitive and to create software efficiently, but for customers with the manufacturing mentality [AL], Agile is a synonym for anarchy and chaos. Knowing the IT supplier practices a nice well-defined methodology with comprehensive planning and detailed design documentation such as MMDIS may be a big benefit in their eyes.

That's exactly the way many plan-driven methodologies survived until now. The only issue with MMDIS is, it is not know on a world-wide scale. Managers usually don't read software methodology books. Usually they find their way through unknown using buzzwords and MMDIS won't probably ring a bell.

---

30 At this point it is important to recall that MMDIS is not restricted to software development but has a wider scope which has not been examined in this text.

31 The author of this article calls a marketing methodology the one which is not to be followed but which is to be announced to be followed for obvious reasons.

32 They don’t realize how disciplined in fact Agile is [AL2]. For example XP is a highly disciplined process which is very demanding in terms of the practitioners.
Bibliography


