Humanization of Work in Project Management

Andrzej Borucki

Faculty of Engineering Management
Poznan University of Technology
ul. Strzelecka 11,60-965 Poznan, Poland

ABSTRACT

The experience gathered in the course of many IT projects ranging from the simplest to the very complex, shows that the key adverse factors affecting the productivity of project teams are: the stress that accompanies designers in the development and implementation of IT projects, poor use of the intellectual capacities of individual project team members, excessive focus on design support tools at the expense of project quality, excessive adherence to a specific method of IT project management, the pressures of time and budget restraints on projects where labor intensity has been underestimated. According to many IT project managers, the treating of people as "spare parts" often jeopardizes project outcome. Managers commonly forget the simple fact that the intellect and personalities of every employee are different and that often neglected in accomplishing project goals by not helping individual staff members to achieve personal satisfaction. My practice shows that the choice of a design method has little effect on design productivity and often reduces it where the system needs continuous customization to ever new client requirements and restrict the operating freedom on individual projects contributing to less innovative outcomes.

Keywords: Project Management, Software Engineering, Requirement Engineering

INTRODUCTION

A study of the effectiveness displayed by our team[1] on software development projects performed between 2000 and 2013 shows that, in project management practice, excessive emphasis is commonly placed on the technical aspects of project management while neglecting the innovation side. We have observed that the use of standard software development procedures does not make a project more cost efficient nor speed up the design process. What it often does, however, is keep project team members from devoting their time to forging friendly relationships with one another. Practice has shown that mutual relations among designers are pivotal for creating an environment which will unlock the intellectual potential of individual project team members and inspire the team's innovative spirit. Software design managers need to recognize the influence of non-environmental factors on designer productivity and select management methods and techniques which will minimize the arduousness of intellectual work and bring out the innovation potential of the design team (Cushman and Rosenberg, 1991). Our observations show also that, regrettably, the present design practice is geared towards getting designers to become as "efficient" as possible and to making entire projects highly effective. This makes design work extremely strenuous and often described as "the dark side of intellectual work". The reason for this lies in underestimating the importance in the design process of

[1] Since 1990, during his employment in PPH Softmar, the author has led numerous deployments of management information systems. Between 1990 and 2001, he and his team developed and unrolled dozens of proprietary applications in logistics and finance.

https://openaccess.cms-conferences.org/#/publications/book/978-1-4951-2102-9

the human factor and, in particular, people's intellectual and psychological dimensions. In searching for ways to make project teams more productive, the majority of project managers choose to use various CASE support techniques in design work and project management. Such tools are viewed as facilitators for the efficient development of complex IT systems which are more functional, reliable and efficient and allow for easier maintenance and system transferability.

The majority of CASE functionalities are designed to help gather knowledge on the project at hand, create a knowledge-base for future reference, build a project management model and develop a knowledge and team management strategy. Despite their widespread application, such tools do little to keep developers from exceeding deadlines or overspending budgets. . Such problems are particularly pronounced in innovative projects where designing is non-linear and involves a sequence of fixed steps performed up to a certain point in the process after which the developers go into feedback loops not envisioned in the initial project model. In non-linear design, the structure of a software development project may be unknown at the outset. The knowledge which has been acquired tends to be used to unify software development procedures rather than create the kinds of ergonomic conditions for software development that are certain to bring out innovation.

Stress - new challenges for humanizing the work of software designers

Uncertainty in project management

Many project managers lack the means needed to protect projects from failure caused by unforeseeable random circumstances. This applies in particular to innovative projects. In specific project circumstances, decision-making depends on one's knowledge of the possible developments. Each decision boils down to the choice of one of a range of possible solutions. That is why, before reaching a specific decision, one needs to assess the impacts of all the possible options. . IT project management allows for defining four situations in which solutions need to be selected which is similar to decision-making theory (Buslenko,1967):

1. Well-defined situations where the impacts of actions result clearly from specific parameters. Such circumstances occur where specific individuals are to be assigned to a project – the project costs will then be calculated on the basis of the base pay and overtime rates of such individuals and using a specific cost allocation method (proportionately, at the start or completion of a project task). In such a case, decisions are said to be made in well-defined situations (spaces).
2. Random situations in which the consequences of decisions depend on parameters which may offer some degree of foreseeability and can be defined by means of probability distribution. Such circumstances may occur in repeated projects when the knowledge repository provides an idea of the time needed to produce specific repetitive project parts – the estimates are based on the time taken by other programmers to develop the same program part in other projects. Experience shows that programmers tend to differ widely in how efficiently they can design the same program segment. In such circumstances, decisions are said to be made in random situations (spaces).
3. Uncertain situations are circumstances occurring commonly in managing innovative projects where the impact of decisions is affected by factors which are uncertain, i.e. factors which may be subject to change and offer no indications as to what specific changes could be expected. In the case of actual IT projects, the choice of resource types and its use is made at the stage of project definition which is when one is not aware of the time to be allocated to the performance of individual tasks. On many innovative projects, decisions on how to order tasks and associate interdependent tasks with one another tend to be postponed. Innovative IT projects certainly meet the definition of projects on which decisions are made in uncertain situations (spaces).
4. Conflict situations in which certain parameters which affect the consequences of decisions made in the course of design work result from progress made on competing simultaneous projects. Circumstances in which designers are simultaneously engaged on multiple projects are common in companies which pursue entire IT project complexes – such circumstances produce conflict space. Under such circumstances, the impacts of decisions differ widely and depend on the circumstances at hand and the responses of parties to conflicts. Decision-making in the conflict space may be described with the language of the game theory which investigates conflict models in which project managers are given a choice from one of many possible strategies.

The consequences of choosing specific design options are often assessed by common sense and precisely this is a common source of trouble. Typically, the selection of impact assessment criteria is followed by a quantitative evaluation of the consequences of choosing a specific project approach.

Information stress

Information stress results from the inability of employees to process excessive input data supplied to them in the course of software development or from not having enough information to complete their work. At the start of the design process, it is difficult to tell which information will be of use and value. Information stress results from the discord between the volume of information provided during software development and the individual processing capabilities of the designers. Such stress can be seen to affect developers with varying intensity at all stages of software design and deployment – it affects all persons involved in creating software as well as its would-be users (who are also involved in the design process). Experience with a great number of IT projects which varied in the degree of their innovation and complexity points to the following key factors which undermine the productivity of designer teams is to stress experienced by designers in developing and implementing IT projects results from:

- their post-go-live accountability for software defects;
- their inability to process a flood of input data supplied during software development or having to complete their assignments based on incomplete information;
- having to cope with gaps in knowledge, especially technological, needed to complete their project,
- changes to software requirements made in mid-project, especially by the prospective user.

Figure 1 shows three key non-environmental factors which affect the ergonomics of software development and which can be influenced by team manager.

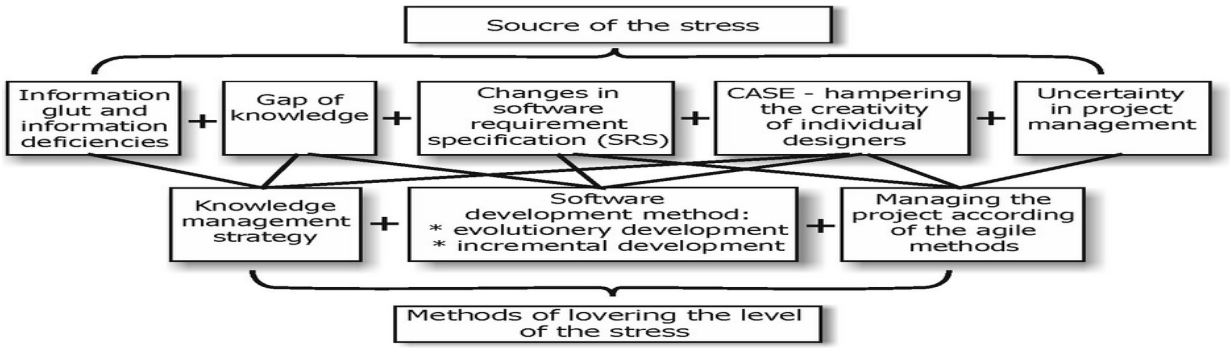

Figure 1. Non-environmental factors affecting key ergonomic conditions for software development.

A number of factors determine an individual's sensitivity to information stress. Such factors include inquisitiveness manifesting itself as hunger for knowledge, temperament, personality and rational intelligence. Information stress may be experienced at various stages of information processing (Ledzińska, 2009), (Hankała, 2009). Excessive or insufficient information at various design stages may affect the severity of the stress experienced by designers. Stress may also harm the working efficiency of developers who are bombarded daily with aggressive information. The resulting information glut may impair their ability to process and select information causing them to make misguided judgments. Information stress may distort cognition and result in delivering products based on false or incomplete assumptions. In extreme cases, it may contribute to project failures. Our observations of dozens of software development projects suggests that designers experience the most severe information stress while identifying and examining functional software requirements. At this stage, conflicts abound due to excessive functional demands made by various groups of would-be users as well as functions left out by designers and the system functionalities unnoticed by future software users.

In designing large systems in projects which go on for months on end, requirements concerning specific functionalities may change considerably over time. This may be due to the persons responsible for selecting software functionalities changing their minds, changes in the effectiveness factors applied on the business parts of IT architecture for which a given software package is being developed and changes in the legal environment of a given

enterprise. The practice of software design is a complex process of compiling documentation to support specific ideas for designing a model of the finished system. Software design documentation usually contains specifications of requirements as well as contextual, behavioral and data models and the documentation necessary for software rollouts and testing. The design documentation should indicate the purpose for which specific software is being developed, its structure and functionality. It should also recommend configurations to ensure proper software installation and offer operating guidance for software users. Software documentation should come in either a simplified or a full format. A good example are web-based applications operated in a dispersed processing environment which require full documentation. In their case, the logical model developed in, for instance, the UML technology, should be captured in five different system model perspectives, i.e.:

- the perspective of sample uses expressed in diagrams of applications and activities (used for system behavior modeling),
- the process perspective developed by means of class and object diagrams (depicting system structure models) and activity diagrams designed to present a model of the states of a given process and the transitions across such states as well as a control flow model,
- the implementation perspective presented by means of component diagrams,
- the implementation perspective presented by means of implementation diagrams,
- the design perspective expressed by means of class, interaction and state diagrams used to model system behavior**.**

For each of the perspectives, developing project documentation requires gathering a huge amount of information in the repositories of various CASE tools. In practice, even where unlimited and equal access to data has been ensured for all designers assigned to a specific project area, it may still be impossible to properly select and classify such data by relevance. It may therefore happen that the available information on functional requirements for the software in development may be mutually complementary but also mutually exclusive, overly general or overly detailed at a given stage of system design. A common problem associated with the use of CASE tools in designing IT systems is to ascribe information to a specific design stage at which it is to be used, which results in data overloads at specific stages. While leading several software design teams in the last two years, I have noticed that demand for access to project knowledge varies from one designer to another. Many have shown a great desire to learn all details of every issue. Others showed more restraint in seeking to obtain a wide range of information, which did not at all mean their performance was in any way inferior. While leading several software design teams in the last two years, I have noticed that demand for access to project knowledge varies from one designer to another. Many have shown a great desire to learn all details of every issue. Others showed more restraint in seeking to obtain a wide range of information, which did not at all mean their performance was in any way inferior.

Methods of lowering the level of the stress

Knowledge management strategy

A key role in developing software is played by the intellectual resources of project teams, i.e. their knowledge. The knowledge used to produce software may be either open or hidden. As knowledge grows, it is placed in repositories available to designers and ready for use on a wide range of projects. Multiple use of the knowledge stored in CASE tool repositories is believed to be critical for rapid software development and expected to streamline the design process. The knowledge held in such repositories concerns software design, implementation and testing. The options for its use are numerous and depend on the knowledge management strategy adopted for a given design project. Three distinctive knowledge management strategies are available for managing software development. These are the knowledge codification strategy , the knowledge personalization strategy and strategy mixt (Jashapara A., 2004), (Borucki, 2011).

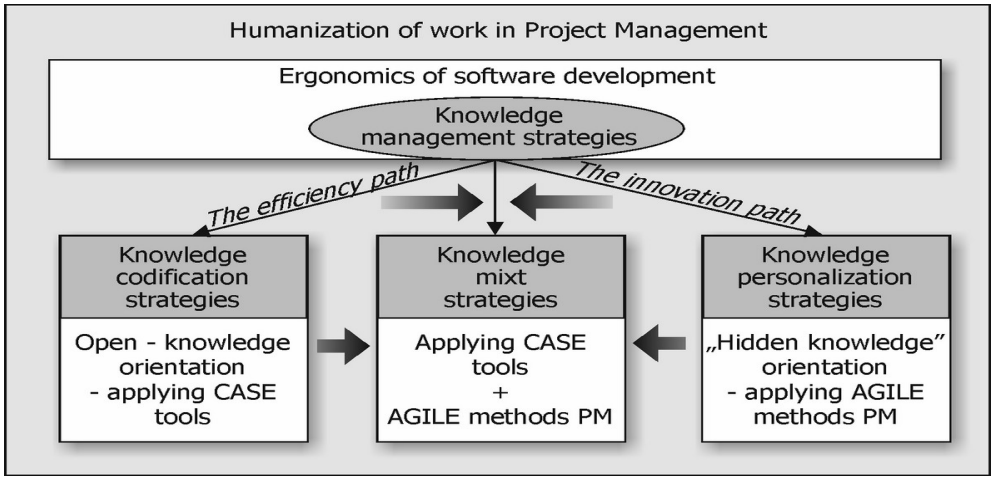

Figure 2. Alternative approaches to the knowledge management strategy.

As proposed by the knowledge codification strategy, project management involves rational gathering of knowledge on technical issues encountered in the design process and inserting it into databases, procedures and the documentation of logical and physical models developed during a project. The use of a knowledge codification strategy in IT project management resembles approaches to repetitive production where the technological process of producing a product follows a prescribed structure and comprises such individual components as operations, measures and items. Each component of the technological process employed to produce a machine part is described in great detail in a database (of standards) in terms of the depletion of physical and financial resources, machine and equipment uptime and labor intensity. Similarly in software development, the primary purpose of the information held in CASE tool databases is to build logical and physical software models suited for repeated use on various projects. The knowledge codification strategy is intended to make the software development process repeatable and base it on well-tested components and development methods. The knowledge codification strategy requires the use of expensive technology to apply CASE tools. It entails high costs of strict compliance with the design and documentation principles stipulated in CASE manuals. The knowledge codification strategy makes it necessary to use CASE tools. Based on two decades of experience in managing IT projects, we have grown somewhat critical of the use of CASE tools (Borucki, 2012).

On the down side, CASE tools undermine designer productivity by contributing to:

- limiting the creativity of software designers;
- preventing osmotic communication among members of the same design team and across multiple design teams assigned to a single project;
- making the design process more rigid by strict adherence to the design methodology imposed by CASE tools;
- compounding project management problems wherever the design process becomes non-linear and random;
- imposing limits on the roles assigned to design team members;
-“thoughtless” work resulting from strict reliance on knowledge provided in the CASE tools database;

An overview of successful IT projects in recent years, particularly those relying on internet technologies, shows that a key to their success is to employ a proper knowledge management strategy which helps bring out and effectively utilize the intellectual potential of individual design team members.

Another approach to project management is to apply the knowledge personalization strategy. In the context of project management, such a strategy places emphasis not as much on design supporting technology but rather on personnel improvement by mutual interaction intended to share knowledge and employ designer creativity in problem solving. Such a mutual exchange of information on problem issues encountered in implementing various project components promotes innovation and mitigates failure risk. Managing projects in keeping with the knowledge personalization strategy does not entail any additional expense.

Ergonomic software development methods

The requirements put to designers are ever more demanding in terms of the time allowed to deliver new software services. Requirement changes are managed in the four stages of change identification, change analysis, change monitoring and change planning. All requirement modifications need to be examined not only for implementation issues but also in terms of the time and cost of their adoption. In a rapidly changing environment, requirements management becomes an essential skill of software engineers (Berenbach et al.,2009). Due to highly changeable business models and shorter software modification time, application developers are forced to resort to adaptive rather than predictive and non-linear methods which allow for the structure of a given project to be partially unknown at project launch as the design project includes feedback loops decisive for the definitions and solutions adopted further into the design process. Such an approach to software development is only possible when knowledge management in the project team follows the knowledge personalization strategy. The strategy aims at osmotic communication which allows exchanges of information among project team members on the basis of equal access. Project team members are granted access to all project information regardless of whether such information is of direct relevance for the design tasks they perform or concerns another set of responsibilities. The assumption is that one can never be sure if a given piece of information will not be needed further in the project and turn out to be helpful in solving a problem which cannot yet be foreseen. If the knowledge codification strategy is applied in software design management resulting in extensive use of CASE tools, volatile requirements will wreak havoc with painstakingly composed project schedules and prevent clients, i.e. future software users, from intervening on an ongoing basis. This is because to intervene in such a manner, the clients would first have to learn the language and detailed rules of using specific CASE tools. The benefits of replacing CASE tools with the knowledge personalization strategy become obvious on very unique and challenging projects conducted to highly changeable software design requirements.

In the practice of IT systems design, two of the many available software development methods are considered to be ergonomically sound in terms of knowledge personalization strategies and incremental development . In both of these methods, project success depends chiefly on the effective use of all creative potentials of individual designers working together and on the creative collaboration with the future software user. Such an approach to working with the prospective system user allows the client to accept the proposed designs much easier as the client is involved in design development by providing its business expertise to contribute to defining requirements and testing individual design components.

Evolutionary development

One method of software development is the evolutionary approach in which a preliminary (working) software version is created and submitted to the user for preliminary approval of the system requirements referring to the basic functional specifications (Sommervile, 2003). At this stage of software development, designers collect feedback from the future user which they subsequently incorporate into the successive intermediate prototypes. The evolutionary development method requires that software specifications, software development and software approval proceed in parallel. A diagram illustrating evolutionary development is provided in Figure 3.

https://openaccess.cms-conferences.org/#/publications/book/978-1-4951-2102-9

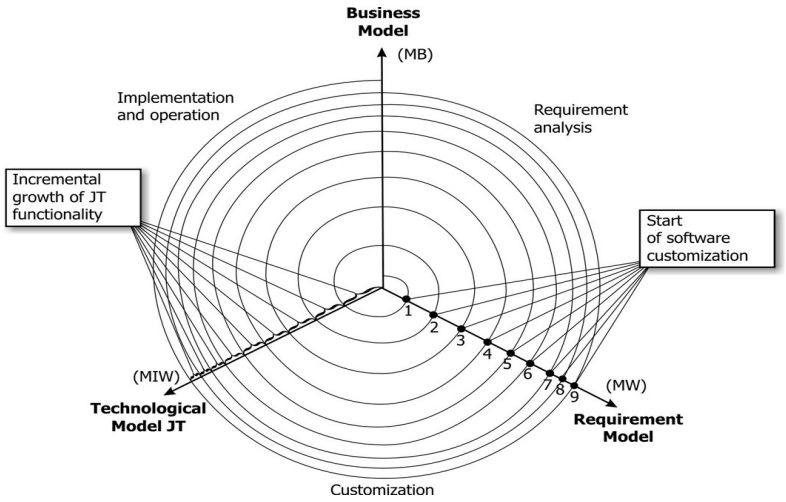

Figure 3. Evolutionary development.(Adopted from Borucki and Pacholski ,2010)

Evolutionary development allows one to accept requirement changes in the course of software development. By providing the software user with intermediate versions, one makes gradual adjustments of the software to meet individual requirements. As the experience of many software projects demonstrates, evolutionary development is well suited for small as well as large programming projects.

Two evolutionary development models have emerged in business practice (Sommervile, 2003):
- in one such model, software development begins with the requirements which are best known to designers as they have been applied on other projects or constitute core functional requirements of vital importance for satisfying the business needs of future software users. The literature describes this sort of evolutionary development as “research-based”,
- the other type of evolutionary development, known as “throwaway prototyping”, entails developing and providing the client with a software prototype to be tested to arrive at the ultimate version of functional requirements. In practice, the client is provided with multiple successive prototypes, some of which are discarded after preliminary assessment. This software development method is used commonly in designing the modules of large software systems.

In both types of evolutionary development, software specifications are developed incrementally to allow the future user to gradually gain insights into designer intentions and respond on whether the functional solutions in the software meet its expectations and comply with the company's business model.
Our vast practical experience proves that the evolutionary software development method reduces the negative implications of asymmetries in development knowledge between system designers and users. Such asymmetries are one of the main causes of failures of business management software projects.

Incremental development

Another ergonomic approach known as incremental development**,** clearly helps create an atmosphere of cooperation which reduces the overall adverse effects of stress resulting from information gaps and the variability of requirements seen on most software development projects (Sommervile, 2003).

A distinctive feature of incremental methods used for software development is that they allow clients to:

- take part in each stage of design,
- defer certain decisions on the functionality of software (or at least selected modules) until they are fully familiar with issues and experienced with previous software increments.

The incremental software development method provides clients with successive “increments”, each of which may well be seen as a project in its own right. A diagram of the incremental development process is provided in Figure 4.

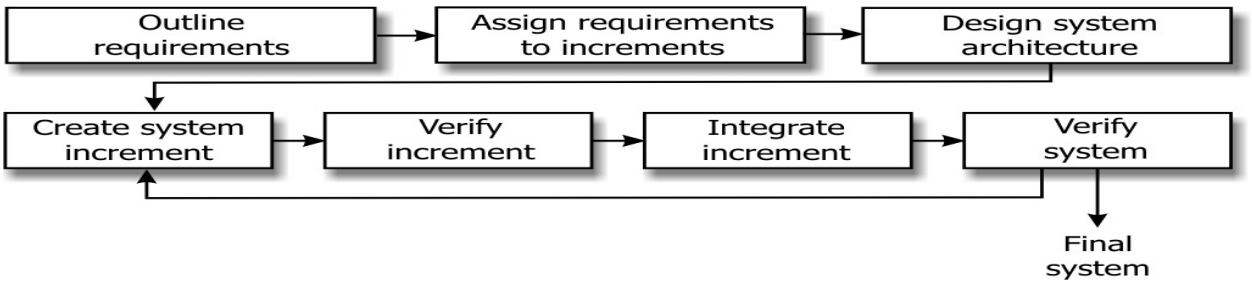

Figure 4. Incremental software development.(Adapted from Borucki and Pacholski,2010).

At the initial stages of software development, the client is expected to show the designers which functional requirements need to be satisfied first with the successive increments. An increment should satisfy a certain orderly version of functional requirements. Every increment is built around a full cascade structure made up of definitions of requirements, the design, implementation, testing, integration and the testing of the complete sum of individual increments which the would-be user has identified as its top priority. The key functional requirements should be delivered in early increments. The designers’ job is to work closely with the client to identify such increments.

The incremental method of software development makes it possible to deliver the parts of the system the client cares most about at very early stages. The client gets to experiment with such system parts to learn more about the system. Clients may also provide their feedback on how satisfied they are with system functionality and how the increments help them perform the requested services and define further increments, i.e. draw up a list of system functions to be delivered in the next increment. To ensure the mutual consistency and integration of individual increments, it is necessary to develop a preliminary system architecture model comprised of three components: a business model, a requirements model and an implementation model. The preliminary system architecture model should reflect a specific level (view) of system abstraction to allow the future user and designer to define individual increments, specify functional requirements and estimate project costs, including the costs of building the technical infrastructure required for software implementation.

Agile software development methods

The best software development method is selected by a number of criteria, the most critical ones being project size, project complexity and the degree of involvement of the future user in software design. The term agile software development describes the ability to produce software quickly in response to fast-changing requirements (Martin and Martin, 2009). Project team leaders prefer the so-called light customization methods:

- whose design is human-factor-oriented as they allow for software engineer creativity ,which are more adaptive then predictive meaning that IT designers go through a *system life-spiral* in the iteration process as they develop successive prototypes and then verify them against updated requirements,
- which are non-linear meaning that the ultimate structure of a programming project may be defined only partially at project launch and fine-tuned through the feedback loops at successive stages of development.

Most IT projects are one-off ventures. The choice of a particular tactic is seen as vital for running them in a structured and consistent manner. . Such documentation helps to document progress (and prepare bills of quantities) as well as to develop designs at the maintenance phase of the software lifecycle. Another anticipated advantage of completing full design documentation was the added capacity to manage human resources, especially at times when unforeseen events arise. Unfortunately, the practice of applying rigid software development rules has killed creativity as processes and tools take precedence over technical solutions and client satisfaction. An Agile Software Development Manifesto was announced on February 11-13, 2001 in Snowbird, Utah, USA. The authors of the Manifesto sought to change the existing rigid rules of forcing software developers to apply a prescribed method for drawing up design documentation in software design practice. The authors of the Agile Software Development

Manifesto devised 12 principles in an attempt to define a paradigm for developing software whose distinguishing feature is the so called agile programming used in its development which differs significantly from the common variety of software development (Martin and Martin,2008) , (Highsmith, 2007). The approach focuses on:

1. **Client satisfaction.** The software designer's top priority is to satisfy the client by delivering, as soon as possible, software that meets the current requirements imposed by the company's business model.
2. **Changeable needs.** One must accept that needs will change even if defined late in the design process. Software designers should not shy away from clients ordering software modifications who then change their minds and redefine previously agreed requirements. The design team should strive to develop a flexibly structured product in anticipation of new demands. The ability to build software in an environment of frequently changing needs is a powerful asset of programming firms that will definitely set them apart from other providers of such IT services. A design team that is open to change will be more likely to recognize the problems and expectations of future software users. Agile software development prolongs software lifecycle. Combined with customization, the approach offers programming firms a substantial revenue potential as software is a product that generates customer loyalty. The longer customers use a given product, the stronger their attachment.
3. **Frequent delivery.** In keeping with the above rule, software developers should divide the product delivered to their clients into increments (by the iterative software development method) which represent successive software versions. Each such version builds on its predecessors. Early versions should focus on the most challenging and controversial functionalities. Frequent delivery of software increments allows designers to verify the extent to which their product meets client needs. This mitigates the risk of project failure and helps tighten the relationship with future software users.
4. **Closer to business users.** The key prerequisite for the success of IT projects is to work closely with future users at each stage of software development. By recognizing business problems and learning about a client's business model early on, designers can define software increments in response to client needs. What is more, business and programmer jargons are "tribal languages", as it were, imbued with technical terminology that stands in the way of communication between the two parties. Close cooperation between programmers and business clients alleviates the problem.
5. **Reliance on talented and well motivated members of the design team**. IT project managers often believe that designer effectiveness depends primarily on their technical skills and working time management. To increase designer effectiveness, managers seek to standardize design procedures and processes by using CASE tools and pressurizing their people to work longer and more efficiently even at the expense of product quality. Such an approach to designer work stems from the fact that many managers graft their human resource leadership practices from production floor. The common claim is that a well-managed design team can be changed in mid-project without compromising project completion (Marco and Lister, 2002).
6. **Talking and "osmotic" communication is the best way to exchange information among design team members.** The authors of the agile software development approach believe that one-to-one conversation is the best way to exchange information within the design team. Documents and other written matter may only supplement oral messages. One should therefore arrange shared space in the workplace for team members to meet and freely exchange information (Marco and Lister, 2002).
7. **The key measure of design work progress is properly working software.** The software delivered to client should come in well thought out increments that contain functionalities needed to support business processes. Such an approach will help assess progress in satisfying the needs of future software users. Hence, design project progress is measured by the number of functions that meet client expectations rather than the size of documentation or the number of source code lines.
8. **Continuous and sustained software development at a steady pace.** Agile designs are suited for frequently changing requirements – their development methodology is adaptive rather than anticipatory meaning that product development is iterative (and follows e.g. the spiral pattern). Each iteration ends with the production of a prototype to be verified against an updated requirements model. The work should continue at a steady pace over a long term to ensure the resulting software meets high quality standards.
9. **Technical perfection.** Agile projects are not linear meaning that early on in design development some of their components are not sufficiently recognized. As a consequence, care for technical details will help develop a source code to the highest quality standards.
10. **Simplicity.** Agile software development is suited for small and simple projects by the rule of "tailor-making" to meet current needs. One should develop customized designs in response to the clients current business needs while remaining open to change and retaining scalability to allow for easy modifications of both the software as well as the system's logic model documentation (e.g. in UML).

11. **Self-organizing design teams.** Successful agile design must allow for the creative working style of each designer and creative client relations. Agile development designers should themselves assign tasks to individual team members and decide how to approach them in the best way. The team members should rely on mutual osmotic communication. This will ensure equal access to project knowledge for each team member. Each designer should be allowed freely to comment on the work-in-progress while taking responsibility for the total team work.
12. **Continuous improvement of organization, operating procedures and team communications.** Agile design requires continuous adjustments of teams to changing client demands. A team's ability to adapt to new working conditions is pivotal for project success.

The authors of the Agile Software Development Manifesto suggesting that the way to effectively manage personnel is to allow designers to adopt an individual working style. Bear in mind the basic fact that each employee has its own unique personality which needs to be recognized in project implementation by ensuring the designers are personally satisfied.

CONCLUSION

Our research has revealed the following leading causes of difficulties in managing innovative IT projects to support management:

1.The non-linear nature of management software design,
2.Inadequate scheduling methods failing to recognize the random nature of certain schedule inputs and uncertainties
3.Asymmetrical access to all project information by project developers,
4.Project managers keeping team members from showing initiative and limiting their roles,
5. Rigid adherence in the design process to the regime of a specific design methodology
6.Applying CASE tools without adopting measures to assess the quality of designers' performance,
7.Hampering the creativity of individual designers.

In order to manage software development effectively, advantage needs to be taken of the knowledge held by each design team member. This is to ensure that the resulting software meets the requirements defined by its future user and arrives on time and on budget.

REFERENCES

Berenbach B., Paulish D., Katzmeir J., Rodorfer A. (2009) , "*Software and Systems Requirements Engineering*" : In Practice. New York: MCGraw-Hill Professional.

Borucki A. (2011), "*Knowledge management in software customization*",J. Lewandowski, I. Jałmużna, A. Walaszczyk (eds.), Contemporary and future trends in management. Wydawnictwo Politechniki Łódzkiej,

Borucki A.,Pacholski L. (2010), "*The human factor in Lean Development*", P.Vink, J.Kantola(Eds).Advances in Occupational,Social and Organiz. Ergonnomics.CRS Taylor&Francis Group.Boca Raton,London,New York.(pp.763-772.

Borucki, A**,** (2012), **"***Factors Adversely Affected the Productivity of Software Designers Applying CASE Tools* ", Advances Social and Organizational Factors, Ed.Peter Vink , CRC Press Taylor&Francis Group ,317-329

Buslenko N.P.i inni (1967), *„ Metoda Monte Carlo*" , Warszawa PWN

Cushman W.H., Rosenberg D.J. (1991), *" Human Factors in Product Design*", Elsevier, Amsterdam,

Hankała, A. (2009), *„Aktywność umysłu w procesach wydobywania informacji pamięciowych*", Wydawnictwo Uniwersytetu Warszawskiego, Warszawa.

Highsmith J**.,** (2007), "*APM:Agile Project Management*", PWN.

Jashapara, A. (2004), "*Knowledge Management*", An Integrated Approach, Peearson Edu cation Limited.

Marco T., Lister T. (2002), *„Czynnik ludzki, skuteczne przedsięwzięcie i wydajne zespoły*", WNT, Warszawa.

Martin R.C, Martin M**.,** (2008), *„Agile. Programowanie zwinne. Zasady, wzorce ,praktyki zwinnego wytwarzania oprogramowania w C#*" ,Wydawnictwo Helion.

Kotonye G., Sommervile J.(1998), *" Requirement Engineering:Process and Techinques*", John Wiley and Sous.

Laplante P. (2009), "*Requirement Engineering for Software and Systems*" ,(1st.ed) Redmond, WA:CRC Press.

Ledzińska, M. (2008), *„Człowiek współczesny w obliczu stresu informacyjnego*" , Wydawnictwo Psychologii PAN.

Ledzińska, M. (2011), *„Nowe czasy-nowe źródła stresu w pracy. Obciążenie psychiczną pracą-nowe wyzwania dla ergonomii*".

ed. by Juliszewski T., Komitet Ergonomii PAN, Kraków
Sokołowska J.,(2005), „*Psychologia decyzji ryzykownych*", WSWPS Academica.
Sommerville I. (2003), „*Inżynieria oprogramowania*",PWN,Warszawa.
Stallman A., Greene J. (2005), "*Applied Software Project Management*", Cambridge", MA: O'Reilly Media.