Grażyna HOŁODNIK-JANCZURA*,
Izabela GOLIŃSKA**

DECISION SUPPORT SYSTEM FOR CHOOSING A MODEL FOR A SOFTWARE DEVELOPMENT LIFE CYCLE

The aim of this paper is to present selected models of a Software Development Life Cycle as a set of possible alternatives. The article also includes the characteristics of IT projects which are used as the basis for selection criteria, according to which an appropriate model should be chosen. These characteristics are divided into two groups; one of them deals with the product, the other one deals with the project. Based on both a literature study and statistical surveys, a list of criteria is derived, to be later applied in the process of developing a knowledge-based system. The rules and search algorithms for selecting the best models are described by a flowchart. Finally, the method of presentation and the interpretation of the results are discussed.

Keywords: *algorithm, knowledge base, sequential model, evolutionary model, IT project, selection criteria, risk, project complexity*

1. Introduction

From an IT project manager's point of view, choosing an appropriate model of a Software Development Life Cycle (SDLC model) is a strategic issue, whereas on a theoretical plane it is a very complex and weakly structured[1] problem. To resolve

---

* Institute of Organization and Management, Faculty of Computer Science and Management, Wrocław University of Technology, ul. Smoluchowskiego 25, 50-372 Wrocław, e-mail: grazyna.holodnik-janczura@pwr.wroc.pl

** A 2009 Faculty of Computer Science and Management graduate.

[1] Structure is a feature of decision-making problems related to the possiblity of constructing a straightforward, quantitative model for decision-making. In the case of weakly structured problems, where many criteria are unknown at the beginning and only elements of theoretical knowledge can be represented by numbers, it is difficult to establish a set of solutions and define decision-making procedures [1, p. 11].

this problem properly, one should employ not only pure knowledge, but also the experience and intuition of managers, aided by expert opinions. Thus, conclusions stemming from investigating both sides of the issue were used to link the characteristics of SDLC models with the characteristics of IT projects; hence the development of criteria for model selection. Despite the fact that the bibliography on SDLC models and the methodology of IT project management is extensive, the problem of choosing the most suitable model for a specific project remains without a straightforward solution. Therefore, bearing in mind the complex nature of the problem examined, one should consider employing a knowledge-based system, in which the work of an expert is aided by a software program that mimics his reasoning. Such systems belong to one of the many subcategories of a broader group of management support systems. Among other subcategories one can mention e.g. Decision Support Systems (DSS), expert systems, or knowledge-based systems [5]. Since these systems share many of their characteristics, the borders between particular categories are not clear-cut. Therefore, the system proposed in this article is considered as an interactive expert system that incorporates a knowledge base.

2. Models of an IT project life cycle

A software development life cycle is a process that includes mutually consistent stages, which allow the complete and successful formation of an IT system and its following use. This process covers the time span from the point of realizing the necessity for creating a system, to the moment of its decommission. Early models dealt mainly with the amount of work needed to create a successfully performing system; elements such as planning, quality control and risk analysis were often omitted in their design. However, models of such life cycles have evolved, so at present these elements are taken into consideration more and more frequently. As a result, integrated models of an IT project life cycle are called models of a Software Development Life Cycle.

2.1. Sequential models

The classic examples of sequential models are the waterfall model and the V-model. Both of them include a description of the approach used, which derives from the division of the software programming process into technologically sequential stages. According to the rule saying that the output of one stage is the input of the

following one, stages need to be completed one after another. In its less rigid version, the waterfall model allows dividing each stage into two parts. The first part is concerned with the technological work corresponding with that particular stage of development, while the second consists of verification and validation. The goal of verification is to confirm that the project under construction is consistent with the prior specifications, while the validation part is aimed at assuring that an adequate product, which here means a product desired by the client, is being built [7], [8].

There are many advantages to the waterfall model. It arranges tasks in an order that facilitates the tracking and monitoring of progress. In its altered version that includes verification and stage-by-stage confirmation, the model shares some features of quality and configuration management. Nonetheless, real-life projects rarely run in a sequential order, and even though one can introduce repetitions into the model, they are considered unnatural and confusing. One of the difficulties that is hardest to overcome when using a sequential model is the fact that at the beginning clients are usually unable to pinpoint all of their requirements. Yet, even if this were possible, alterations to the requirements frequently occur at various stages of the construction process, especially if the system is advanced. Abiding by the rule of a sequence of actions also hinders arriving at a solution to this problem. Another drawback of the sequential model is its inability to present and consult the client with a working version of the software, because it becomes available relatively late in the course of development. This may lead to a situation in which a serious error in a project is discovered only at the end of a cycle, which will substantially increase the total cost of the project. Additionally, working in such a manner results in keeping certain members of the project team unoccupied; the consequence here is again an increase in costs.

Despite all these deficiencies, sequential models are still utilized, mainly due to the relative simplicity of managing a project that uses them. The waterfall model is perfectly applicable for small or middle-sized projects with specific and stable requirements [7, pp. 29–31].

2.2. Evolutionary development models

On one hand, the market pressure emphasizing the need to issue products faster and faster, forces producers to release programs as early as possible, even if the only version available is limited. On the other hand, it is widely known that requirements undergo changes, even in the course of programming, and that while the requirements concerning the core of the system are exact from the start, the minor details are usually unclear. Therefore, it seems beneficial to adopt an evolutionary approach, which should be understood as a model in which the statuses of the various stages are intertwined in reference to the elements of the software under develop-

ment. Using an evolutionary model, the issuing of a product proceeds in certain stages; the functions of an initially limited first version are upgraded with each update of the product [7, pp. 35–42].

The group of evolutionary models consists of numerous prototyping models and their modifications, such as e.g. the spiral model and its variant the Win-Win model, the parallel model, the incremental build model and the Component Object Model (COM). The characteristics of a classic spiral model are presented below.

The spiral model

The spiral model of the software development process was defined by BOEHM [2]. The beginning of a project is placed at the center of the spiral; its development follows the unwinding of the spiral. Each cycle represents one stage of a project under development. The requirements in the center are not well defined, yet details are added with each rotation of the spiral. For instance: the outcome of the first cycle is product specification, the outcome of the next one might be a prototype and the following outcomes are constantly upgraded versions of the software designed. The cumulative cost of a project rises with the length of the spiral. The model is considered evolutionary, since it includes both recurrence, which is characteristic of a prototypical model, and the regularity of a classical model. In addition, the scope of action of an evolutionary model is extended by a risk management phase.

As a matter of fact, the spiral model is merely a package containing other models, therefore no particular way of handling the sequence of phases is imposed. Even the number of stages, usually between three and six recurring in every cycle[2], is variable. The keywords associated with the spiral model are: objective determination, planning the project, quality control and risk management. All the abovementioned factors may have a significant impact on successfully completing the project and meeting the deadline.

The spiral model appears to be a realistic approach to large and complex projects. The software designer has enough time to fully comprehend the clients' requirements, while the clients can supervise the construction of the product, seeing if what is being built meets their expectations. However, there are certain shortcomings to this model. First of all, it demands intense involvement of the staff in managing the project, in particular in terms of risk management. A second considerable drawback is the limited utility of this model in designing systems that are meant to function as simplified ver-

---

[2] Boehm's spiral includes four phases: the first one, in the upper left corner, involves determining objectives, searching for possible solutions and their limitations, the second one (going clockwise) identifies and resolves risk and assesses the alternative solutions; in the course of the third one the program is developed (this area would overlap with the classical waterfall model to a large degree). The last, fourth, phase includes either planning the next stage or iteration. Each full completion of a spiral cycle involves complementary, cross-sectional activities, such as quality control or configuration management [2].

sions with a lower standard of quality, when the deployment of the final product is staggered in time [6, pp. 288–289].

2.3. Model selection problem

It is difficult to give a straightforward answer to the question: which model is the best? Each of them is established upon different foundations, applicable to some types of projects, but not to all of them. For instance, in the case of some models, such as the waterfall and the Rapid Action Development (RAD) model, one presumes that all the requirements concerning the software are available in the preliminary stages of the project. Whereas in the case of the spiral, the Win-Win and the prototypical models, one assumes that the initial specification might be simply incomplete and that it might not be possible for the clients to state their requirements for the project, or that they may not be fully understood by the system developer. Therefore, it is generally assumed that one should familiarize oneself with the basic parameters characterizing a given project prior selecting a suitable IT project SDLC model. Only after that can one begin deciding what models would meet specific project and product requirements.

When it comes to projects that are aimed at designing life-critical systems, whose main feature is high reliability, one should obviously apply a model ascertaining the development of a reliable system. In the case of a pressing deadline, one should probably opt for an RAD model. However, which model should one choose if there is not too much time, but there are high demands concerning reliability? Therefore, the model selection issue is becoming a multicriterion problem. Solving it is up to the project manager, whose task is to select the factors describing the software, whose analysis would be the most beneficial for the project.

3. Product and project characteristics as criteria for model selection

The literature on IT project SDLC models includes many factors characterizing an IT project that influence selection of an SDLC model. This paper will present the best known of these, taking contemporary knowledge as a reference point that aids the creation of a list of criteria for developing a computer-aided DSS for model selection. The criteria were divided into two groups: product, and project criteria. The criteria concerning software, in other words: the product group, include: the type of information system, the size and complexity of the software, system architecture, modularity

and level of module integrity, the variability and clarity of user requirements, or generally speaking – the quality of software. The criteria concerning project management, in other words: the project group, include basic project dimensions such as: planned timetable and cumulative costs of the project, resource characteristics, project risk, types of users according to computer skills and fluency with computer systems.

3.1. Product characteristics

Type of information system

Information systems used by various institutions and companies can be divided into three categories [4, pp. 18–22]. The first category comprises commonly used systems that facilitate organization. Their main objective is to help employers in fulfilling various everyday tasks. Office software such as MS Word and MS Excel, or teamwork aiding programs such as Lotus Notes and Novell Group Wise are examples of such systems.

The second category of information systems comprises so called domain systems. This group can be further divided into two subgroups. The first of these consists of systems that are aimed at improving the efficiency of business processes in e.g. customer service or financial management. Examples of such systems are specialized Fixed Assets Management Systems (FAMS), or Customer Relationship Management (CRM). The second subgroup consists of systems created to aid creative activities, e.g. design or decision support processes. Elements characteristic of this subcategory are highly-advanced applications, such as systems meant for industrial designers and architects (Computer Aided Design, CAD), or systems aiding technological planning processes, designed for industrial engineers (Computer Aided Process Planning, CAPP).

The third category of systems is constituted by integrated management systems. This branch of systems is being steadily introduced into the field of applications that integrate the various levels on which companies function. The most advanced of such systems also cover strategic issues at the functional level of an organization. The best known examples are ERP[3] integrated management systems, such as the SAP[4] software family, Oracle Applications, IFS Applications[5]. Due to the large size of both such integrated

---

[3] Enterprise Resource Planning (ERP) is a term describing a class of integrated IT systems. Systems of this kind aid managing a company in the fields of e.g. finances, accounting, HR, wages, supplies, production planning.

[4] Systems Applications and Products in Data Processing (SAP), a company founded in 1972, is a world-renowned leader in software providers, developing systems for businesses working in various sectors of the economy.

[5] Industrial & Financial Systems Applications (IFS) are a set of integrated IT solutions, comprising around 70 modules, eg. in finances, production, supply chain management, customer relationship management, project and resource management.

systems and the companies to which they are applicable, implementing all the modules at the same time could result in serious disruption in the functioning of the company. Therefore, the implementation of such projects is usually achieved in stages.

To select an SDLC model for any of the categories characterized above, one should take into consideration its characteristics. For instance, models with a clearly distinguished maintenance stage seem to be the most appropriate for the first category, while models which emphasize the management of change and iterative cycles seem to suit the second category best. The third category seems to work best using models combining the characteristics of the first two types with the additional possibility of applying modular fragmentation in the design and implementation of software.

Software size and complexity

An important, though difficult activity to be done at the very beginning of an IT project is software sizing. This problem is usually solved by the use of various assessment methods based upon mathematical models, by brainstorming, or by the Delphi method. Depending on the accuracy of the outcome, the assessment data serve as an approximation of the size of the planned software6. Research shows that there is a straightforward relationship between software size and the amount of work needed for its development [7, pp. 172–174]. Its exponential form points to the necessity of dividing large software projects into small parts, to reduce their complexity and ease project management in general. Therefore, the models that seem to be the most suitable in this case are incremental or evolutionary models, or, in the case of software carrying out complex algorithms, it would be a formal transformation or prototyping model.

Computer system modularity and the level of module integrity and complexity

The possibility of designing a modular framework for computer systems definitely simplifies work on an IT project. Problems related to the complexity of a computer system are frequently resolved by dividing the project into smaller parts, which are to be worked on by individual project teams. Nevertheless, one should bear in mind that the number of modules should be optimal, i.e. not too big, since the bigger the number of modules, the more difficult the final integration becomes[7]. The integration process tends to be time consuming. However, the benefits of modularity are priceless, mainly because the complexity of a large project is dispersed over smaller tasks, each tackled individually.

---

[6] Amongst the most popular measurements are measures of length, e.g. Lines of Code (LOC), or functionality measures such as Function Points (FP), Use Case Points (UCP), or Cosmic Function Point (CFP), previously known as the COSMIC functional unit (Cfsu).

[7] Another reason for this is the increasing number of problems in integrating the communicating parts, since the number of connections between parts is $n(n - 1)/2$, where n is the number of parts.

Quality standards

Quality is a combination of measurable features which determines the rating of a particular product, usually by comparing it to certain established standards. Software quality can be defined as "the property of meeting clearly stated requirements concerning software development and the unspoken expectations for each type of professionally designed software" [8, p. 202]. This definition underlines three significant aspects of quality. Firstly, the basis for measuring quality should be clearly defined requirements – if a product does not meet them, it is of poor quality. Secondly, software should be developed with regards to established standards. Lastly, quality software should not only meet the overt requirements, but also unspoken expectations, such as the protecting against the possible effects of program closure and data backup in such cases. Therefore, in the act of selecting an SDLC model one should also take quality control into consideration and check if it is ascribed to each stage of the cycle or not.

3.2. Project characteristics

Planned timetable and budget/cumulative cost of the project

The time frame and budget are the most basic characteristics of any project and, as recently published research claims, the success of a project depends largely on them [9]. With long-term projects involving high levels of investment it is necessary to prepare a stage-by-stage plan for project realization. A meticulously constructed timetable that incorporates milestones for a project definitely increases the chance of successful closure. Nowadays, software is frequently developed within a prior stated time frame, with the option of an extended deadline; this is characteristic of the incremental build model. When full software implementation is not possible at the appointed time, the most vital aspect of functionality is chosen. In such a situation, clients are able to put this part of the computer system under their scrutiny and decide whether the project should be continued or not. This usually means an increase in the time frame, which in turn results in an increased budget.

Project resources

Under the term resources, one should differentiate between human and technical resources. Human resources are the group of people who are narrowly specialized in a designated stage of software development and who are able to implement the project successfully. In this case, the group is called a project team, and includes (depending on needs): management, analysts, software designers, computer programmers, software testers, and specialists in the fields in question. At each stage of development, depending on the SDLC model used, each member of the team works at a different

intensity. Apart from a quantitative approach to human resources, it is important to take into account such factors as experience in the implementation of similar projects or in utilizing certain technologies. If a project team works with an unfamiliar technology, project risk increases, as well as the duration of the project in general, due to the possible necessity of staff training or extended search for solutions to problems.

The second type of resources are composed of technical resources, which comprise not only the necessary hardware, but also tools – software – together with access to the chosen technology. Both the available and required level of resources can substantially limit this choice or simply suggest the best model for the project e.g. RAD model – large project team, COM (*Component Object Model*) – available components, the prototypical model – CASE tools.

Project risk

Risk accompanies all IT projects. This notion concerns the possibility and probability that something will go wrong. For an IT project, this basically means a lack of certainty regarding the deadline and hindrances in the course of a project's realization. Risk is characterized by two markers: uncertainty, in the form of the threat of what might arise from a future event and the loss incurred from a future event that turns out to be negative.

Risk cannot be entirely removed from IT projects, yet it can be successfully managed. Research indicates the weakest elements of an IT project. Selecting an adequate SDLC model significantly decreases risk levels. For instance, if a project team is assigned to design an innovative system and the client's requirements are unclear, one should choose a prototypical model, which at the initial stages helps to detect possible difficulties that may hinder work later in the course of project development. Other models, such as the spiral model, include risk analysis at each step of the project, and thus they are of considerable use in the implementation of high-risk projects.

4. Expert opinions

To obtain data on the issue of model selection, surveys were sent to project managers working in selected IT companies. Amongst other issues, the interviewees were to answer the following questions: "What is the best way to select an SDLC model?" and "What are the most vital criteria in selecting a model?" However, despite a couple of interesting answers, the sample turned out to be too small to formulate any far-reaching conclusions.

To sum up the poll results, one could state that the obtained answers provided confirmation for the theoretical plane of the issue under investigation. An analysis of the basic project dimensions proved that delays in deadlines go hand in hand with delays

in finances. Also, a significant regularity was noted, related to task prioritization. Apparently, when used within an incremental build model, prioritization facilitates the development of a computer system remarkably.

We also attempted to establish a list of potential factors that could influence the selection of an SDLC model. The results show that in the case of a prototypical model connected with RAD, the incremental build, or the Build-And-Fix model, the most significant factors are the time frame expected for the completion of the project, and user priorities, followed by the size and complexity of the project. The spiral model proved to be successfully applicable not only to large and medium-sized projects, but also to small ones. The theoretical arguments for selecting the RAD model, namely time limitations, were also confirmed, while the incremental build model, which appeared to be the most popular one, worked perfectly for larger projects characterized by a more complex architecture. This last model gained favor with project managers because it can be used by an inexperienced team, although they pointed out the inconvenience deriving from the necessity of frequent meetings with the users.

5. The concept of a knowledge-based system

The goal of developing an expert system with the aid of artificial intelligence is to come up with results that would help to establish which SDLC models are particularly suitable for a particular IT project. In other words, obtaining one clear answer was not the aim, but rather obtaining guidance in choosing from a known set of models. The model selected should suit the characteristics of a given project the best. To make this task possible to resolve, the characteristics of a project should correspond to the criteria set out in the selection of model; the criteria should be stored in the knowledge-base of the system. Then, to obtain information about a suitable model, the users of such a program would have to determine the characteristics of their projects by answering a set of questions asked by the system. The full functioning of the system can be depicted by a hierarchy of windows in the interface of the application, as arranged in two panels: the first accessible to the administrator, the second accessible to the decision maker.

From the user's point of view, the system is accessible through a double-layered structure (figure 1.). The first layer is to be used by an expert and serves as a tool for parameterizing the system according to the models considered, criteria and management of the question facility. It remains invisible to common users. Only the second layer, enabling the choice of a model is accessible by decision makers. With the use of an appropriate interface, decision makers can enter the known characteristics of their projects. Obtaining information from decision makers happens as a simple question-answer process. A successful and apt selection of an SDLC model, which is the outcome of the system, depends on appropriate definition of the expert, criterion layer. Separating the criterion

layer from the question layer exempts the decision-maker from the necessity of identifying various characteristics of models and of examining their influence on successful project implementation. The wearisome stage of determining criteria and associating them with questions is completed by an expert, who in this system is called the administrator. The presence of a definable criterion layer offers ample opportunities for calibration of the decision-making system. With the use of such a tool one can accurately render the dependencies related to the selection of an SDLC model in real project development.
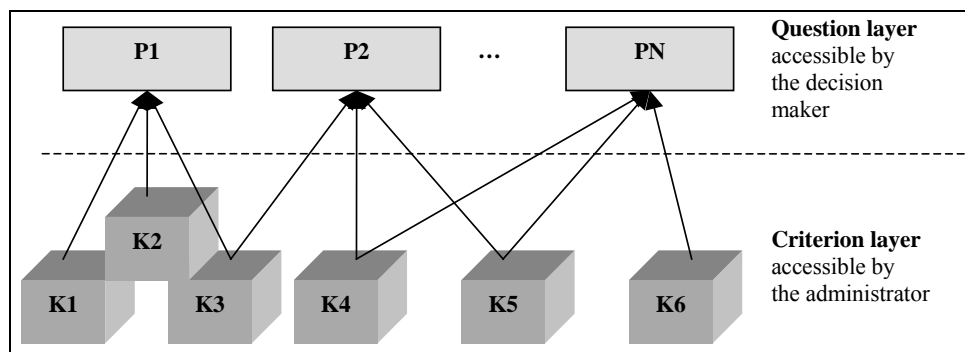
Fig. 1. Abstract, layered division of the decision-making system

6. Knowledge-base

Containing all the information concerning SDLC models in a database that can be accessed in the decision-making process is difficult, since it is not only defined by quantitative measures, but mainly by description. In order to resolve this problem, several simplifications and assumptions were made to aid the design of a database and to make the project clear and logical. This is of most importance when it comes to concluding the process, since this is based upon the facts stored in the database. As a result, a base was designed that contains knowledge about the links between the characteristics of models and projects. This became the basis for the selection algorithm.

The database comprises of two parts. The first one stores the set of available models and information about their features; exact values or ranges of values if necessary and text values are ascribed to these features. The second part maintains the links between the questions, answers and the conditions upon which the conclusion of the process works.

The initial version of the system included 11 models, as follows: the waterfall model, the V-model, the prototypical model, the RAD model, the incremental build model, the spiral model, the Win-Win model, the parallel model, the Component Object Model (COM), the formal transformation model and the Build-And-Fix model.

The application possesses an additional asset, which is storing the history of the answers given by users and keeping a table of results for the models, which is updated after each answer. Apart from being able to access the history of the answers, the users are also able to return to the previous question whenever they want.

7. Application parameterizing –
permanent knowledge-base updating

Application parameterizing means filling the database with appropriate components. In the initial version of the system, the list of components was based on both research in the literature on the field and expert knowledge. The data entered can be browsed, deleted, edited, and updated. All changes (add, delete, edit) are automatically saved in the system database. To do this, one has to open a module named *Parameterizing – administrator panel*. This part of the application is designed for expert users having considerable experience in IT project management. With the use of their practical and theoretical knowledge, such users will be able to enrich and improve the existing database.

Fig. 2. The "Edit module parameters" window with an open "Edit" option
and a drop-down list of criteria

New models can be added by filling out a table in the upper part of the "Edit module parameters" window (figure 2). Not only has each model to be characterized in

detail, but also it is required to establish links between each of the models and the selection criteria. The significance of a particular field should be highlighted as the "fulfillment_level_parameter". This is used to calculate the score assigned to each model, which is vital in assessing the levels of significance of particular criteria to a particular model. It was assumed that this field takes values from 1 to 100. Let us use "team experience" as an example criterion. The text values assigned to it are "little," "medium," and "large," e.g. programmers with much experience may be a necessary criterion for using some of the models (the aforesaid field attains a value of 100 in such a case), while they might only be a recommendation for others (field value – 50). To sum up, this field enhances the sensitivity of the system, hence upgrading the mechanism of concluding the process.

8. Algorithm for selecting a model

The algorithm for selecting a model is a heuristic ranking algorithm (figure 3). The outcome of its operation is a list of models together with the ascribed score, in order from the highest to the lowest scoring model.

The principles of model selection are stored in the *Conditions* table. Employing a chosen condition to answer a given question links the criteria to the model. The initial list of criteria included: software size, membership of a life-critical system, software purpose, clarity of requirements, main system function, project duration, financial resources, team experience, user experience, user contact. Calculating the score for models which were not excluded from the set of possible solutions is according to a defined equation. Two cases were taken into consideration: the existence of, or the non-existence of a link between a condition and the parameters of a given model. If a link exists, the score is calculated using:

$$m_{ij} = m_{ij-1} + \frac{kw + pw}{2} \cdot \frac{sp}{100} \qquad (1)$$

where:

m_{ij} – the value of the *model_points* field in the *Models* table, which includes the previous sum of points collected by model i in stages 1, 2, ..., $j-1$ of the selection process,

kw – the value of *criterion_weight* in the *Criteria* table, which stands for the weight of the criterion to which a given condition is referring,

pw – the value of *question_weight* in the *Question* table, which stands for the weight, or in other words, the importance of the question,

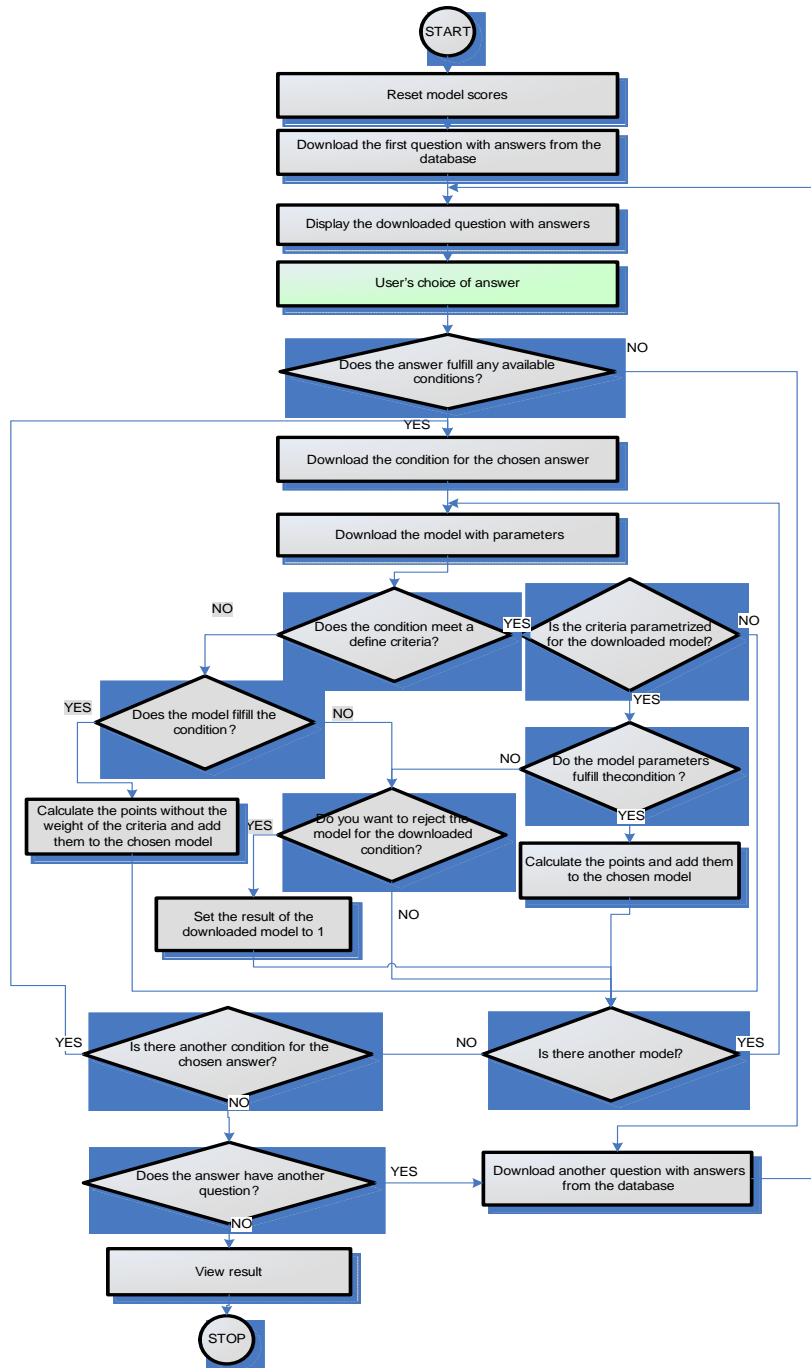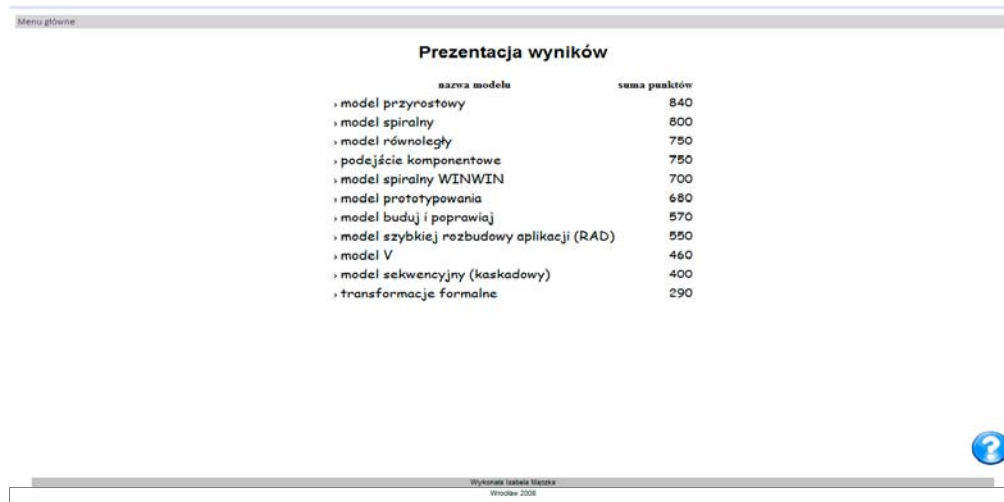sp – the value of *fulfillment_level_parameter* in the *Parameters* table.

Fig. 3. Algorithm for selecting an SDLC model – a flowchart

Points are assigned to those models whose parameters fulfilled the conditions of individual answers to the given questions; their number depends on the average weighted by the "weights" of questions, and the values according to each criterion ascribed to a given condition, diminished by the value of the *fulfillment_level _parameter* parameter.

In the case of the non-existence of a link, which is true if the given condition cannot be connected to any criteria and the *fk_criteria_id* in the *Conditions* table is empty, points are calculated according to the following equation:

$$m_{ij} = m_{ij-1} + pw \tag{2}$$

where:

m_{ij} – the value of the *model_points* field in the *Models* table, which includes the previous sum of points collected by model i in stages $1, 2, ..., j - 1$ of the selection process,

pw – the value of *question_weight* in the *Question* table, which stands for the weight, or in other words, the importance of the question.

9. Presentation of results

Once the answers have been given, the application shows the results on the screen (figure 4). The list presents all the models entered into the *Models* table, together with

Fig.4. The "Results" window

the score they were given by the algorithm. The best SDLC model for a given project is supposed to be the one with the highest score. All the other models are presented with their scores, so that the choice of the highest-ranked model is not imposed on users. On the contrary, they can make their final decision upon examining the whole decision-making path and analyzing the full model characteristics presented by the system.

The "Decision-making path" window (figure 5.) shows all the answers selected by the user. The upper row gives the model's number, the lower row shows how the total score for each model changed according to how the questions had been answered. Thanks to this, users can track the ways in which their answers influenced the final score of a model, both on the spot and after obtaining the results.
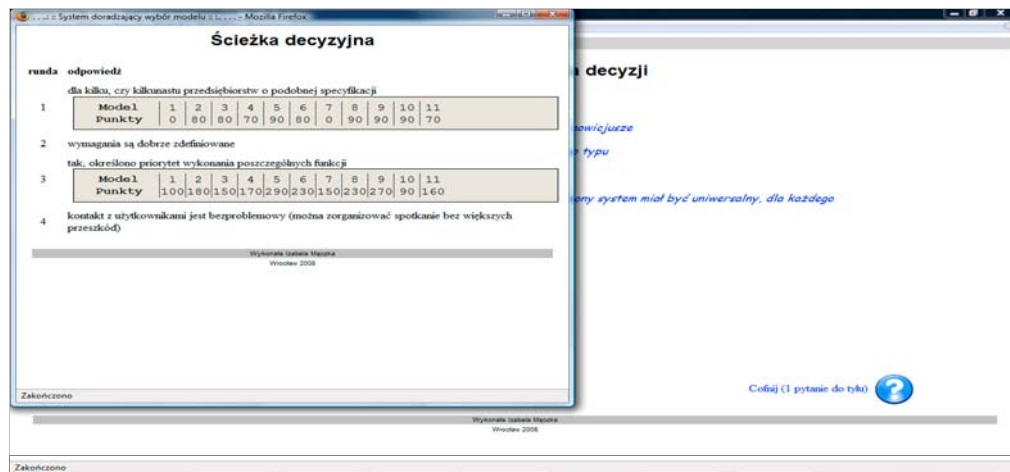

Fig. 5. The "Decision-making path" window

The result of implementing this algorithm cannot be considered as entirely objective, for it is highly reliant on the weights ascribed to the individual criteria, as well as the questions. In addition, scores are assigned to the models according to the level of criterion fulfillment. All these values are subjective and can vary depending on the priorities of the decision-maker, who has the power of setting them according to his preferences.

10. Summary

Selecting an SDLC model can be compared in many ways to the specification of user requirements; the more data gathered and examined, the higher the chances for

successful completion of the project. Just as the specifications of user requirements are vital in the stages of design and computer system development, so can the knowledge and regulations which constitute the basis for SDLC model selection determine the success or failure of a given project.

To sum up, selecting an appropriate SDLC model is a complex and a challenging task, which requires not only broad theoretical knowledge, but also consultation with experienced expert managers. Therefore, the computer application presented should be perceived as the first step towards building a system that could be applied in practice; the possibilities for its development depend on the activity of its users. The flexible construction and permanent parameterization method used in the system makes it a multi-tasking tool for decision support, which not only gives decision-makers the opportunity to learn, but also allows them to participate in a system's development.

References

[1] BIELECKI W.T., *Informatyzacja zarządzania*, Państwowe Wydawnictwo Ekonomiczne, Warszawa, 2001.

[2] BOEHM B.W., *A Spiral Model of Software Development and Enhancement*, IEEE Computer, Maj 1988, 61–72.

[3] CADLE J., DONALD Y., *Inżynieria oprogramowania, Zarządzanie procesem tworzenia systemów informacyjnych*, Wydawnictwo Naukowo-Techniczne, Warszawa, 2004.

[4] FLASIŃSKI M., *Zarządzanie projektami informatycznymi*, PWN, Warszawa, 2006.

[5] KISIELNICKI J., SROKA H., *Systemy informatyczne biznesu*, Agencja Wydawnicza „PLACET", Warszawa, 2001.

[6] KOLBUSZ E., OLEJNICZAK W., SZYJEWSKI Z., *Inżynieria systemów informatycznych w e- gospodarce*, Polskie Wydawnictwo Ekonomiczne, Warszawa, 2005.

[7] PRESSMAN R.S., *Inżynieria oprogramowania. Praktyczne podejście do inżynierii oprogramowania*, WNT, Warszawa, 2004.

[8] SOMMERVILLE I., *Inżynieria oprogramowania*, WNT, Warszawa, 2003.

[9] www.infog.com/articles/interview-Johnson-Standish-CHAOS