Expert Systems with Applications 39 (2012) 406–418

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A rule-based approach for estimating software development cost using function point and goal and scenario based requirements

Soonhwang Choi[a,1], Sooyong Park[b,1], Vijayan Sugumaran[c,d,*]

[a] DMC R&D Center, Samsung Electronics, Suwon, Gyeonggi-do 443-742, Republic of Korea
[b] Department of Computer Science, Sogang University, Seoul 121-742, Republic of Korea
[c] Department of Decision and Information Sciences, School of Business Administration, Oakland University, Rochester, MI 48309, United States
[d] Department of Service Systems Management and Engineering, Sogang Business School, Sogang University, Seoul 121-742, Republic of Korea

ARTICLE INFO

Keywords:
Project management
Requirements triage
Cost estimation
Function point
Goal
Scenario

ABSTRACT

Function point is a method used to measure software size and estimate the development cost. However, for large complex systems, cost estimation is difficult because of the large number of requirements expressed in natural language. In this paper we propose a rule-based approach for estimating software development cost in the requirements analysis phase. It combines goal and scenario based requirements analysis with function point based cost estimation. In our proposed approach, Context Analysis Guiding rules, Data Function Extraction Guiding rules, and Transaction Function Extraction Guiding rules have been developed to identify function points from text based goal and scenario descriptions. These rules are established based on a linguistic approach. The contribution of the proposed approach is to help project managers decide which requirements should be realized.

1. Introduction

Developing software systems that meet stakeholders' needs and expectations is the ultimate goal of any software provider seeking a competitive edge. To achieve this, we must effectively and accurately manage our stakeholders' system requirements: the features, functions, and attributes they need in their software system (Davis, 1993; Şen & Baraçlı, 2010). Once we agree on these requirements, we can use them as a focal point for the development process and produce a software system that meets the expectations of both customers and users. However, in real world software development, there are usually more requirements than we can implement given stakeholders' time and resource constraints. Thus, project managers face the following dilemma: how to select a subset of the customers' requirements and still produce a system that meets their needs (Karlsson & Ryan, 1997)?

Software requirements triage is the process of determining which requirements a product should satisfy given the time and resources available (Davis, 2003). The practice of triage increases the likelihood that a product will meet customers' needs, and thus contributes significantly to the economic impact of that product on the company's bottom line. Yet despite the potential benefits of requirements triage, not much research has been conducted, and the descriptions that do appear are brief (Davis & Zweig, 2000; Leffingwell & Widrig, 2000; Wiegers, 1999; Yourdon, 1997). This is because triage is a difficult task, fraught with political and financial dangers—politically dangerous because both technical and marketing personnel claim the tasks as part of their responsibility; financially dangerous because a mistake could trigger a major loss of revenue. Davis (2003) has studied the requirements practices of approximately 100 companies and organizations over 25 years. He reports that one of the 14 key recommendations is to contain "Cost". Boehm (1981) and Boehm and In (1996) argues that project managers should be able to manage the impact of changing requirements on software cost and schedule. Therefore we need to identify the scope of project and develop an appropriate plan for project at the initial phase of software development.

Recently, some proposals have been made for estimating cost from requirements. Larvet and Vallée (2002) have defined an estimator based on the information available at the requirements stage, mainly the text of requirements. They advocate a set of textual metrics that could be predictors; it includes four distinct metrics, namely, TNW, NKW, AUTO and MANU. It is possible to quantify the complexity of a project through simple linguistic metrics that are directly issued from the requirements. Auer, Becker, Rauber, and Biffl (2005) proposed a transparent way of visualizing the similarity of use cases without the need for explicit data collection; an implicit analogy metric is obtained by using textual similarity. Lavazza and Valetto (1981) have presented a case study that

* Corresponding author at: Department of Decision and Information Sciences, School of Business Administration, Oakland University, Rochester, MI 48309, United States. Tel.: +1 248 370 2831/+82 2 705 8845.

E-mail addresses: soonhwang.choi@samsung.com (S. Choi), sypark@mail.sogang.ac.kr (S. Park), sugumara@oakland.edu (V. Sugumaran).

[1] Tel.: +82 2 705 8928.

aims at quantitative assessment of the impact of requirements changes, and estimation of the cost of development activities that must be carried out to accomplish those changes. They adopted a very simple approach to requirements quantification: they just counted the labels placed on textual requirements for identifying them and tracing them to design and code elements.

Although there are several different approaches for estimating software project efforts (Li, Xie, & Goh, 2009; Park & Baek, 2008; Pendharkar, 2010), few of them are actually applied successfully in typical industrial environments. One of the main reasons is that each estimation technique usually relies on its own unique way that is subjective. They need to be combined with general cost estimation method like function point. However, they are not very conducive to be formally used with general cost estimation methods, and do not provide any methodological guidelines for how to estimate cost from requirements. Finally, these approaches do not directly help project managers in selecting a sub set of the requirements for implementation.

In this paper, we propose a rule-based approach to count function point from textual requirements through goal and scenario based linguistic approach (Kim, Park, & Sugumaran, 2004, 2006). Thus, our main objective is to develop a methodology for counting function point from textual requirements. Two characteristics of the proposed approach contribute to the achievement of this objective.

First, textual requirements are specified in terms of goal and scenario. We adopt goal and scenario based approach for requirements engineering from prior work (Kim et al., 2004, 2006). The second characteristic of the approach is the concept of extraction rules to count function point. These rules guide the user to count function point. There is a key notion embodied in the extraction rules. The idea is that the goal and scenario at the interaction level describes the interaction between user or external application and the target application. It includes data for interaction, which can be used to derive *data functions*, and behavior for data processing that leads to *transaction functions*.

The rest of the paper is organized as follows. Section 2 describes the related work regarding the basic concepts of both goal and scenario approach and function point. In Section 3, our approach to count function point from textual requirements in terms of goal and scenario is discussed using a case example. Then, Section 4 describes the validation of our approach. The last section concludes the paper and describes future work.

2. Prior work

2.1. Goal and scenario based requirements analysis

Goal and scenario based requirements analysis is used to elicit and analyze requirements before function point is counted. After requirements are specified in terms of goal and scenario, function point can be calculated based on goal and scenario. In this paper we adopt the goal and scenario approach discussed in Kim et al. (2004, 2006), which is briefly described below.

The goal and scenario approach has been used to elicit initial requirements and to refine requirements from a higher level to a lower level (Dardenne, Van Lamsweerde, & Fickas, 1993; Kim, Park, Sugumaran, & Yang, 2007; Kim et al., 2004, 2006; Rolland, Souveyet, & Achour, 1998). The goal and scenario approach provides multiple abstraction levels which help in the separation of concerns in requirements elicitation. We use four abstraction levels, namely, business, service, interaction and internal level. The aim of the business level is to identify the ultimate purpose of a system. At this level, the overall system goal is specified by the organization or a particular user. The aim of the service level is to identify the services that a system should provide to an organization and their rationale. At the system interaction level the focus is on the interaction between the system and its agents. The internal level focuses on what the system needs to perform the interactions selected at the system interaction level. A goal is generated at each level and scenarios are created to achieve that goal. Goals at lower level are derived from scenarios at higher level. This mechanism supports the elicitation of requirements through goal and scenario, and helps to refine the requirements.

We also use goal and scenario authoring rules. A Goal is authored as a template $\langle Verb + Target + Direction + Way \rangle$, where Verb is an active verb, Target is a conceptual or a physical object, Direction is either source or destination, and Way is the way in which the goal is to be achieved. In general each goal is expressed as a simple sentence with '*Verb*' and '*Target*' parameter as mandatory. Sometimes 'direction' and 'way' can be omitted. For example, the goal, 'Withdraw cash from the ATM' is represented as follows: '$(\text{Withdraw})_{\text{Verb}}$ $(\text{Cash})_{\text{Target}}$ $(\text{From the ATM})_{\text{Dir}}$'.

The scenarios capture real requirements since they describe real situations or concrete behaviors, and goals can be achieved through the execution of scenarios. Thus, scenarios have their goals, and typically, goals are achieved by scenarios. In other words, just as goals can help in scenario discovery, scenarios can also help in goal discovery. As each individual goal is discovered, a scenario can be authored for it. Once a scenario has been authored, it can be explored to yield further goals. All scenarios should be authored using the following format:

$$\text{`}Subject : Agent + Verb + Target : Object + Direction \\ : (Source, Destination) + Way\text{'}.$$

The expected scenario prose is a description of a single course of action. This course of action should be an illustration of fulfillment of the goal. It should describe the course of actions that are expected, not the actions that are not expected, impossible, and not relevant with regard to the problem domain. Although this format may be perceived to be a bit simplistic, it is sufficient to proceed with modeling goals and scenarios. The indirect objects can be filled in the slot of *Direction*. For example, 'Tom gives me a book' can be rewritten, according to our rules as, '$(\text{Tom})_{Agent}$ $(\text{gives})_{Verb}$ $(\text{a book})_{Object}$ $(\text{to me})_{Direction}$'. The 'Direction' and 'way' are optional in a scenario.

2.2. Function point

Function point is a method for measuring software size which was developed in 1979 (Albrecht, 1979). The method is widely used and adopted as an international standard, namely, ISO14143-1 (ISO-IEC, 1998). It is also adopted by ministry of information and communication in Korea (MIC, 2004). The IFPUG (International Function Point User Group) was established in 1984 and has published FP CPM (Function Point Counting Practice Manual) (IFPUG, 2000). Functions of a software system consist of the following components according to IFPUG's Counting Practice Manual:

- **ILF (Internal Logical File):** data or control information maintained through one or more elementary process of the target application
- **EIF (External Interface File):** data or control information referenced through one or more elementary processes
- **EI (External Input):** an elementary process to maintain an ILF or alter the behavior of the application
- **EO (External Output):** a process that presents information to user through processing logic
- **EQ (External inQuiry):** an elementary process for presenting information to the user through retrieval of data or control information from an ILF or EIF