Chapter VII

Requirements Prioritisation for Incremental and Iterative Development

D. Greer, Queens University Belfast, UK

Abstract

The problems associated with requirements prioritisation for an incremental and iterative software process are described. Existing approaches to prioritisation are then reviewed, including the Analytic Hierarchy Process, which involves making comparisons between requirements and SERUM, a method that uses absolute estimations of costs, benefits, and risks to inform the prioritisation process. In addition to these, the use of heuristic approaches is identified as a useful way to find an optimal solution to the problem given the complex range of inputs involved. In particular genetic algorithms are considered promising. An implementation of this, the EVOLVE method, is described using a case study. EVOLVE aims to optimally assign requirements to releases, taking into account: (i) effort measures for each requirement and effort constraints for each increment; (ii) risk measures for each requirement and risk limits for each increment; (iii) precedence constraints between requirements (where one requirement must always be in an earlier or the same increment as another); (iv) coupling constraints between requirements (where two or more must be in the same increment); and (v) resource constraints (where two or more requirements may not be in the same increment due to using some limited resource). The method also handles uncertainty in the effort inputs, which are supplied as distributions and simulated.
using Monte Carlo simulation before carrying out the genetic algorithm operations. In addition to handling uncertainty, EVOLVE offers several advantages over existing methods since it handles a large range of factors. The overall implementation of the method allows the inputs to be changed at each iteration, and so better fits reality where requirements are changing all the time.

**Introduction**

In any given project, requirements arise from various stakeholders. These stakeholders may be users, developers, project managers, business managers, or other categories of people affected by the system. In the case of new software applications, there are typically a large number of requirements, some of which are essential, others desirable, and some relatively unimportant. For existing applications, there will be a backlog of new requirements, potential fixes, and enhancements, again with differing priorities. In both cases it is usually impractical to implement all requirements simultaneously because of the cost involved, staff limitations, and market or user pressures to have the software implemented. Thus some form of prioritisation is necessary. The output of this process will depend on the effort required for each requirement against the overall effort available, the value arising out of delivering certain requirements at a given time, the risks incurred by delivering or not delivering given requirements, and on dependencies between requirements. In addition to this we have the preferences of business and developer stakeholders, who may be of different levels of importance and may be geographically dispersed with different viewpoints about what should be delivered and when. For some of these stakeholders a given requirement may be essential to the success of the product; others may believe that it might even damage the success of the product. In between these extremes some stakeholders may hold the view that a requirement is unimportant but hold no objection to it being included (Davis, 2003). Overall this is a complex problem, which is very difficult to solve to the satisfaction of all concerned. Management of requirements including priority assignment has been identified as a key success factor for commercial enterprises (De Gregorio, 1999). In the case of bespoke software development, there may be a small number of stakeholders, but in the case of commercial off-the-shelf software there may be hundreds or thousands of stakeholders (Regnell, Host, Natt och Dag, Beremark, & Hjelm, 2001).

**Problem Discussion**

In recent years there has been an increasing recognition that an incremental approach to software development is often more suitable and less risky than the traditional waterfall approach (Greer, Bustard, & Sunazuka, 1999a). This preference is demonstrated by the current popularity of agile methods, all of which adopt an incremental approach to delivering software rapidly (Cockburn, 2002). This shift in paradigm has been brought
Teaching Agile Software Development Quality Assurance
www.igi-global.com/chapter/teaching-agile-software-development-quality/5074?camid=4v1a