Failure to Launch: Scope Creep and Other Causes of Failure from an Actor-Network Theory Perspective

Samiaji Sarosa, Accounting Department, Atma Jaya Yogyakarta University, Yogyakarta, Indonesia
Arthur Tatnall, Victoria University, Melbourne, Australia

ABSTRACT

One of the main causes of delayed and failed information systems project development is scope creep. The increasing number of features demanded by stakeholders to be built into the applications within a fixed time limit is a recipe for failure. This article looks into the process of a web application development failure, where scope creep was deemed as the main cause. An in-depth look into the time line of the project also reveal another cause, which was the failure of the application itself along with the platform (hardware and software) to actually execute the software. It is believed that an Actor-Network Theory framework is appropriate to analyse this case where a number if both human and non-human actors were involved. Data for this research was collected using participative observation. An analysis was conducted to find patterns of negotiations and communications between all the stakeholders during the design process. Actor-Network Theory was used to explain the power plays between actors. A model was constructed showing all the actors (stakeholders) and how the interplay among them developed.

Keywords: Actor-Network Theory, Failure, Requirements Triage, Scope Creep, Stakeholders, Systems Developers

INTRODUCTION

Scope creep is one of the problems of information systems project development (Avison & Fitzgerald, 2006). In his model of Requirements Triage, Davis (2005) proposes that a system requirements specification is a balancing act of three axis, namely resources (usually represented by financial constrains), technical (usually represented by the development team which means required technical effort to build the requirements) and the customer (who usually represents demand for software features). Turban and Volonino (2012) argue that a systems development project is constrained by three factors, which are time (similar to customer), budget (similar to resources) and scope (similar to technical). Where requirements keep growing during the project, this scope creep means that more time and effort is needed to build the systems. More time is usually translated into increases in budget. At the end this could mean a runaway or failed project (Brooks, 1995). In time-constrained development, scope creep is something that is most undesirable for developers.

DOI: 10.4018/ijantti.2015100101
This article analyses a web application development project that ran over time and over budget due to scope creep. The power play of the application developer team (technical aspect), managing director (financial aspect), and president director (customer aspect) were viewed from Requirements Triage (A. M. Davis, 2005) and Actor-Network Theory (Callon, 1999; Latour, 2005; Law, 1999). It is not only why but also how those three triage factors worked and were played through by these actors that is analysed.

SCOPE CREEP AND REQUIREMENTS TRIAGE

Scope Creep is defined as any additional requirements arising during the course of a software development project (Nurmuliani, Zowghi, & Fowell, 2004; Thakurta, 2013; Zowghi & Nurmuliani, 2002). Scope creep is a specific type of Requirements Volatility where the additional requirements are added instead of changed or removed. Any requirements changes that occurs throughout the development process will likely affect the completion of the project. Additional resources (likely including human, technical and financial resources) are then needed, and this will also affect the time needed to finish the project (Brooks, 1995; A. M. Davis, 2005). The Project Management Body of Knowledge (PMBoK) considers that project scope is a serious issue and has a whole section on Project Scope Management and the problems of scope creep (Project Management Institute, 2013). This includes discussion on the collection of project requirements, defining project scope, creating a Work Breakdown Structure, verifying scope and controlling scope.

Although scope creep is considered undesirable by most developers, the reality is that scope creep will often emerge during the development process and is often inevitable (Khan, 2006). Any change in the users’ business needs, changes to the external environment, or even only changes in users’ minds would justify the need to add additional features (A. M. Davis, 2005; A. M. Davis, Nurmuliani, Park, & Zowghi, 2008) and managing scope creep becomes an essential task within a software development project (Thakurta, 2013; Thakurta & Ahlemann, 2011; Zowghi & Nurmuliani, 2002).

Scope creep is not something that is unusual in any engineering of information technology project, including software engineering. Good project management is required to keep the project on the right course towards completion. All the project’s stakeholders need to understand the impact of scope creep toward project completion and also at the end of the project, the functionality of the software produced. Having to understand the impact of any change in requirements the stakeholders can then decide if the changes were needed and justified. However, different stakeholders have different views on what changes are needed. The project manager needs to try to achieve compromise and consensus among the stakeholders.

Davis (2005) proposed a Requirements Triage as a way to balance the different views. A Requirements Triage assumes that in a software development project there are at least three factors to be considered. These are the number of requirements to be built, the desired or available time and the available budget. Requirements Triage tries to balance those three factors by selecting the requirements to be included in the software released in the desired time frame.

Any change in requirements, in this case additional requirements, would most likely require changes in the completion time (or release time) and additional budget. A compromise could be made by only building the reasonable requirements within a specific time and budget. Adding more resources (which translate into additional budget) into the project to finish the software with additional requirements in time, or maybe change in the release time to accommodate the additional time needs to incorporate additional requirements.