

Science of Software Changes

Takuya Katayama

Japan Advanced Institute of Science and Technology

katayama@jaist.ac.jp

Software is subject to change. Changes come from many sources. It has to change as its infrastructure has been changed. It has to change as its functionality has to be augmented. It has to change as new algorithms are found which will increase its performance. There are a lot of reasons why it has to change and only software which can stand for the change can survive. Changes which are gradual and put it into a desirable form is called evolution. This kind of change is essential in constructing a complex software system. Really, it could only be constructed as a result of repeated changes applied to the initial system.

Changes are thus common in software development and much work has been done so far producing many important ideas and concepts. Still, changing software reasonably and effectively is not an easy task. This is by the lack of leading principles for the changes and the inherent difficulties of the changes. If we can find them our understanding and practices of changing software will be greatly advanced.

The author has introduced a seemingly fundamental concept for scientific treatment of software change problems has been introduced, which is called

evolutionary domain. This is a set of objects together with an order relation, called an evolutionary relation, and a difference operator on them. In this talk, I am going to investigate on several current practices and methods for software evolution on this proposed basis, such as components and patterns in OO methodology, feature/collaboration/aspect-oriented development, and MDA approach which is a current highlight of UML based development.

Despite the above methods and practices, software evolution is still hard. It is due to the inherently difficult problem of anticipating changes. Anticipation of changes is usually hard and needs understanding of the world where software is used. We need to know what kind of specifications or requirements for software are permissible there and how they are described. Also we need to know how to define their difference to make one to evolve to others. This is where the evolutionary domain concept could be applied. As an example, Object Logic, which is built on a higher order logic HOL, is introduced for describing an object-oriented world and for showing how evolutionary relation could be defined on thus defined world.