# From reuse library experiences to application generation architectures

Stan Jarzabek
Department of Information Systems and Computer Science
National University of Singapore

**Abstract**

Reuse through application generators has been successful in area of programming language systems. Can reuse practice in other domains benefit from these experiences? Companies usually start reuse at code level, by building a library with reusable code modules. As company's understanding of the domain and experience in reuse grow, reuse may also occur at design and requirement specification levels. We found that systematic modeling of commonalties and variations across systems makes it possible to convert experiences gained during component reuse into application generation solutions. Identifying common and variant structures in a domain is a core activity in building application generators. Domain modeling offers guidelines for representing and studying commonalties, but usually we do not study variations in any systematic way. Understanding variations comes with experience and, due to lack of guidelines and explicit representation, depends much on smartness of a software developer. We can, however, capture and study variations in the same way as we capture and study commonalties. We use a modeling notation that caters for variations. In the paper, we describe modeling methods and a framework that we use to progress from composition-based reuse to, more productive, application generators. The framework reflects our experiences gained in three projects on language processing systems that realized such a transition. We generalized the framework to make some of the concepts useful in the domain of business systems.

## I. Introduction

We analyzed experiences from three projects on language processing systems, a domain where reuse has been particularly successful. We generalized our findings and formulated a reuse implementation framework. We further refined the framework to reflect specific needs of the business system domain. Our framework facilitates transition from composition-based reuse to application generators. We propose modeling of commonalties and variations across a family of software systems in a domain to make such a transition possible. In the paper, we describe experiences and the thinking process that led us to formulating the reuse framework.

The object of domain analysis [1,15] is to identify common features in an application domain. By common features we mean reoccurring requirement patterns, business rules, fragments of software design, data models, objects, abstract process descriptions, modules of code, etc. Common features are made available to programmers either in the form of a library of reusable components (composition-based reuse [3]) or are built into an application generation system (generation-based reuse). Software systems in a given application domain share common features, but we often observe that there are also variations in features across systems [6,9,12]. These variations occur at levels of user requirements, design and implementation. In [9], we described notations for modeling commonalties and variations at levels of user requirements, design and implementation, across a family of similar systems.

Companies normally start reuse at the code module level. At that time modeling should begin. In component-based reuse, modeling variations increases reuse potential as one can easier recognize and understand valid reuse contexts and ways in which software can be customized. It also allows one to avoid doing the same customization process all over again. When experience with reuse techniques and understanding of a domain grow, reuse may also occur at the design and requirement levels (fig. 1). After some time, a component specification model and customized components for reuse in context of different software systems can be studied to identify opportunities for automation. A generic design architecture is developed based on the component design model, while the scope of variations is determined from component customizations. Formal structures developed in component reuse can be gradually transformed into application generators that offer higher level of software process automation and higher productivity gains. In the remaining part of the paper, we describe modeling conventions and implementation guidelines for such a reuse scenario. Then, we discuss two projects based on which we formulated our reuse framework. Finally, we explain how we generalized reuse experiences gained in the programming language domain to make our framework useful in the business system domain.

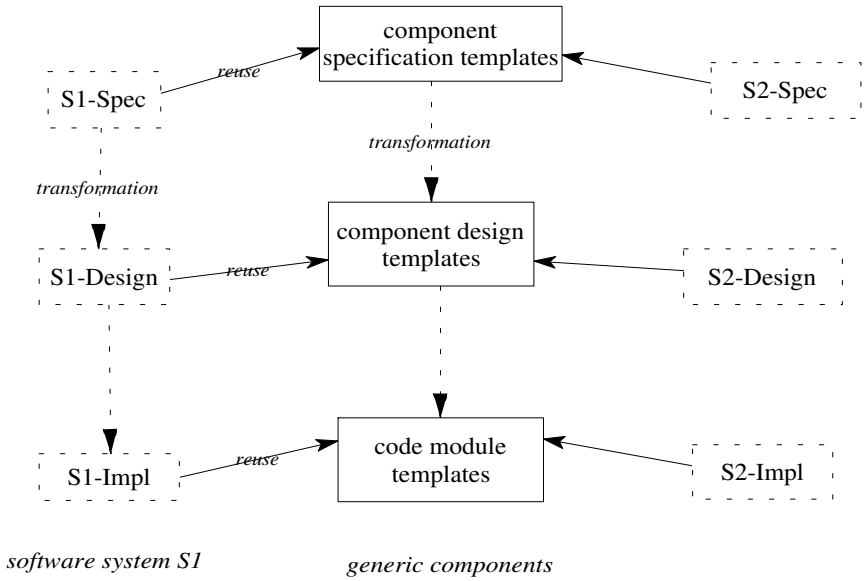


**Fig. 1 Generic library components and their customizations**

## II. Relation to other work

This work was influenced by Parnas papers on design of program families [13] and on using information hiding to design reusable software [14]. We believe application domain knowledge is instrumental in effective reuse [1,2,4,5,11,12,15]. Software models described in this report are based on templates [10]. Synthesis [12] describes domain-specific reuse based on the concept of families of software systems. Like in our approach, Synthesis advocates systematic analysis of commonalties and variabilities. In specific domains, application generators offer more productive solutions to reuse problems than composition-based reuse [2,3,4,6]. Design principles for application generators are discussed in [2,4]. GenVoca model [2] identifies similarities between application generators from different domains and demonstrates how generation approach can be applied to component libraries.

## III. Reuse library structure

Components of a reuse library are features and architectures. *Features* describe meaningful concepts in an application domain. An *architecture* may include many features, combined into a building block underlying a family of similar software systems. A reuse library contains generic and customized components (fig. 2a). Generic components model commonalties, while customized ones model specific systems.

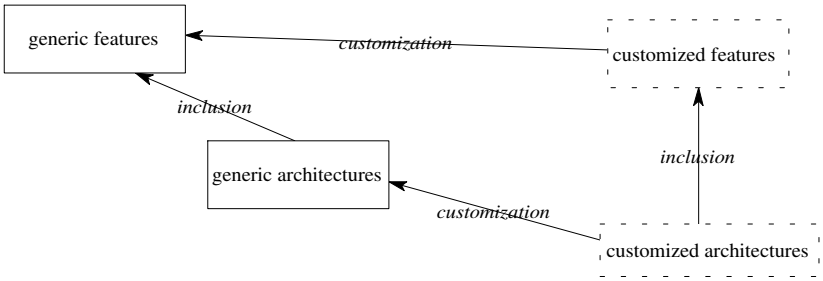


**Fig. 2a Components of a reuse library**

Customization and inclusion of components occurs at levels of specification, design and code (fig. 2b). We use template method [10] to represent components in a library. Accordingly, we have three types of templates, namely specification, design and code module templates.

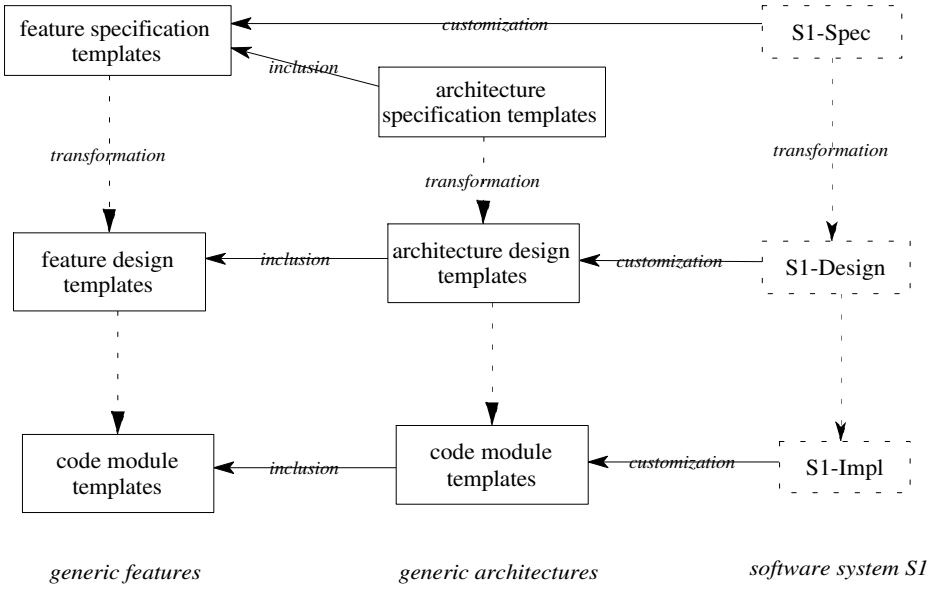


**Fig. 2b Components of a reuse library**

Before we explain the details of a reuse library, we shall outline our global scenario for implementing reuse in a company. First, a company determines the level of its current reuse practice. Then, a general reuse library model of fig.