An Architecture-Centric Approach to Coordination

Sunny Wong

(Advisor: Yuanfang Cai)
Department of Computer Science
Drexel University
Philadelphia, PA, USA
sunny@cs.drexel.edu

Abstract—Empirical studies show that mismatches between design and organizational structures may cause expensive inter-team communication costs. However, prevailing design models and metrics are not designed for identifying independent, globally distributable task assignments, and do not account for organizational structure. Our research objectives are to develop design-centric theories, models, and measures to: (1) minimize the communication needs among developers, and (2) to accurately predict communication needs for performing maintenance tasks. To reach these objectives, we propose to develop design-centric theories and models to identify independent tasks from a design, and to predict coordination requirements from historical change-coupling.

Keywords: software architecture; collaboration; project management; global software development

I. INTRODUCTION

As software continues to grow in size and complexity, software architects often manage this complexity by decomposing systems into modules that can be developed and maintained independently and in parallel. However, simply dividing software into components without considering organizational structure does not guarantee parallel, globally distributable development. Empirical studies show that dependencies between modules often reflect communication needs between their developers, creating a mirroring of design and organizational structures. Coordination overhead caused by these inter-dependent modules can hinder distributed software development and maintenance. The problem is that prevailing design models and metrics are not designed for identifying independent modules for parallel work, and do not account for organizational structure.

Although the two prominent modularity theories (information hiding principle [1] and design rule theory [2]) emphasize the importance of designing software for independent task assignments, prevailing models and metrics are not designed for identifying parallel tasks and cannot quantitatively assess how well a design supports concurrent work. Studies on open source projects reveal that while some software architectures and modularizations support large groups of distributed developers, others do not. Empirical studies [3] reveal a strong correlation between socio-technical congruence and both software quality and productivity. Yet we lack predictive models and measures to manipulate this socio-technical relation and maximize task parallelism.

Developers working on dependent modules often coordinate with each other to avoid inconsistencies, errors, etc. It is important to predict the coordination needs of developers, in order to address problematic coordination issues (e.g. different languages, cultures, timezones). Empirical studies suggest that logical dependencies, which are derived from revision history, are more accurate at predicting coordination requirements than syntactic dependencies. However, we lack a formalization of logical dependencies that accounts for a temporal dimension in order to improve the accuracy of prediction. In particular, logical dependencies do not account for the evolution of software architecture and changes in modularity. That is, as a design evolves over time and the structure of dependencies change, events from the distant past of revision history may no longer reflect the current system's state.

Therefore, we propose to research and develop:

1) a theory with supporting model and measures for identifying independent tasks from a design, and assessing how well a design supports concurrent work.
2) a formal theory generalizing logical dependencies that includes a temporal dimension, for predicting coordination requirements.

II. APPROACH

This section details our proposed program of research. First, we describe a theory and supporting model for identifying, from a software design, independent tasks, which can be assigned for parallel development and maintenance. Then, we discuss a theory and dependency model for predicting coordination requirements.

A. Design Rule Hierarchy

To assist projects managers in identifying the modules of a design that would allow for parallel, independent work among globally distributed developers, we propose designing an algorithm for automatically identifying modules based on the structure of shared design decisions. Our theory on how to identify parallel tasks is based on the design rule
theory [2]. The key insight we make is that a design rule (DR) is a decision that once made, allows its subordinate decisions to be made in parallel. If we can identify the DRs and their subordinate decisions in a design, then that can lead to identifying tasks for concurrent work.

Based on this observation, we propose a design rule hierarchy model, which can be automatically computed from an augmented constraint network (ACN) design model [4], to reveal design rules and their subordinate design decisions. The DR hierarchy partitions a design into a sequence of layers \( L_0, \ldots, L_m \), and partitions each layer into a set of modules. Modules in a layer \( L_i \) do not depend on each other, and only depend on modules in layers \( L_j \) where \( j < i \). Hence, each module represents a task and layer represents a set of tasks that can be assigned for concurrent work. Additionally, since ACN modeling presents a steep learning curve for software practitioners, who are often trained to use UML. To address this issue and support the adoption curve for software practitioners, we propose developing techniques for converting prevailing design models into ACN models.

We recently have designed an efficient algorithm for constructing the design rule hierarchy [5]. Using small software systems, we have shown through manual investigation that the DR hierarchy correctly identifies the design rules of a design. Using 19 months of developer mailing list data from the open source Apache Ant\footnote{http://ant.apache.org/} project, we have shown that when developers are concurrently working on modules in the same layer of the hierarchy, the probability of communication is significantly less than otherwise. And when developers are concurrently working on the same module or dependent modules from different layers of the hierarchy, the probability of communication is significantly greater than otherwise.

B. Stochastic Dependencies

Due to the mirroring of design and organizational structures, we can predict coordination requirements by determining the software elements that are to be modified. Once we know the software elements that are likely to change, we need to find the owners of those elements. That is, we need to know the experts that a developers would likely go to in order to seek advice for modifying those elements. However, in resolving a modification request, managers may not know a priori the set of elements a developer will change. To address this issue, a change scope prediction algorithm can be used to determine the elements likely to change.

Recent studies suggest that logical dependencies are more accurate at predicting change scope than syntactic dependencies. To address the problem that logical dependencies are informal and do not account for the evolution of software architectures, we propose a formalized generalization of logical dependencies based on probability theory, which we call stochastic dependencies. To link design and organizational structure, we propose extending the ACN design model to include a binary ownership relation, which associates software elements to people, and a team set, which models various ways people can be organized to implement a system (e.g., two people may not be put on the same team due to differences in time zones).

Let \( E = \{ e_i \}_{i=1}^N \) be the set of software elements and \( T^{(e)} = \{ t_i^{(e)} \}_{i=1}^M \) be the sequence of revision history transactions that contains the element \( e \in E \) (i.e., \( \forall t_i^{(e)} \in T \), \( t_i^{(e)} \subseteq E \wedge e \in t_i^{(e)} \)). We can view \( T^{(e)} \) as a stochastic process, where each \( t_i^{(e)} \) is a discrete random variable with domain \( 2^E \). Let \( M^{(e)} \) be the length of \( T^{(e)} \). Given two elements \( a, b \in E \), let \( X^{(a,b)} = \{ X_i^{(a,b)} \}_{i=1}^{M^{(a)}} \) be a stochastic process of binary random variables such that \( X_i^{(a,b)} = 1 \) iff \( b \in t_i^{(a)} \). We define the stochastic dependency \((b, a)\) at time \( \tau \) as \( \Pr(X_i^{(a,b)} = 1) \). Unlike some existing dependency types, in which a dependency either exists or does not, our stochastic dependency exists on a continuous range \([0, 1]\).

We base the computation of a stochastic dependency \((b, a)\) on a prevailing model for high order Markov chains, such that \( \Pr(X_i^{(a,b)} = j_0) = \sum_{i=1}^k \lambda_i X_i^{(a,b)} \) where \( \lambda_i \in [0, 1] \) and \( \sum_{i=1}^k \lambda_i = 1 \). Every time \( a \) and \( b \) occur together in a transaction, we gain evidence that they are dependent and this evidence contributes to the predicted probability. Unlike logical dependencies, we only consider the latest \( k \) transactions to account for architecture evolution. By using decreasing \( \lambda_i \) values, we can further account for changes in design and weigh more recent transactions more heavily than older transactions.

III. CONCLUSION

To address the problems that prevailing design models and metrics are not designed to identify independent modules for parallel work, and do not account for organizational structure; we propose to develop architecture-centric theories and models to identify independent tasks from a design, and to predict coordination requirements from historical change-coupling.

REFERENCES