Sostware Engineering for java/ dot net

Check following Projects ,also check if any spelling mistakes before showing to your Guide:

1. Governing Software Process Improvements in Globally Distributed Product Development.

Continuous software process improvement (SPI) practices have been extensively prescribed to improve performance of software projects. However, SPI implementation mechanisms have received little scholarly attention, especially in the context of distributed software product development. We took an action research approach to study the SPI journey of a large multinational enterprise that adopted a distributed product development strategy. We describe the interventions and action research cycles enacted over a period of five years in collaboration with the firm, which resulted in a custom SPI framework that catered to both the social and technical needs of the firm's distributed teams. Institutionalizing the process maturity framework got stalled initially because the SPI initiatives were perceived by product line managers as a mechanism for exercising wider controls by the firm's top management. The implementation mechanism was subsequently altered to co-opt product line managers, which contributed to a wider adoption of the SPI framework. Insights that emerge from our analysis of the firm's SPI journey pertain to the integration of the technical and social views of software development, preserving process diversity through the use of a multi-tiered, non-blueprint approach to SPI, the linkage between key process areas and project control modes, and the role of SPI in aiding organizational learning.

2. Analyzing the Friendliness of Exchanges in an Online Software Developer Community.

Synopsis:

Many online communities struggle with conflicts - e.g. between newcomers and elders - at some point. In July 2012, the Stack Exchange organization attempted to assess the overall “niceness” of the Stack Overflow community by rating the “friendliness” of 7,000 comments made on the site over a 4 year period. We performed a deeper examination of the comment dataset published by Stack Exchange. We find a high degree of comment repetition in the Stack Overflow database and suggest some simple heuristics that may help in automatically identifying unfriendly comments, providing managers of developer communities with simple means that could counter hostility.

3. Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development.

Synopsis:

This study is a quasi experiment comparing the software defect rates and implementation costs of two methods of software defect reduction: code inspection and test-driven development. We divided participants, consisting of junior and senior computer science students at a large Southwestern university, into four groups using a two-by-two, between-subjects, factorial design and asked them to complete the same programming assignment using either test-driven development, code inspection, both, or neither. We compared resulting defect counts and implementation costs across groups. We found that code inspection is more effective than test-driven development at reducing defects, but that code inspection is also more expensive. We also found that test-driven development was no more effective at reducing defects than traditional programming methods.

4. Measuring Code Quality to Improve Specification Mining.

Synopsis:

Formal specifications can help with program testing, optimization, refactoring, documentation, and, most importantly, debugging and repair. However, they are difficult to write manually, and automatic mining techniques suffer from 90-99 percent false positive rates. To address this problem, we propose to augment a temporal-property miner by incorporating code quality metrics. We measure code quality by extracting additional information from the software engineering process and using information from code that is more likely to be correct, as well as code that is less likely to be correct. When used as a preprocessing step for an existing specification miner, our technique identifies which input is most indicative of correct program behavior, which allows off-the-shelf techniques to learn the same number of specifications using only 45 percent of their original input. As a novel inference technique, our approach has few false positives in practice (63 percent when balancing precision and recall, 3 percent when focused on precision), while still finding useful specifications (e.g., those that find many bugs) on over 1.5 million lines of code.

5. A Decentralized Self-Adaptation Mechanism For Service-Based Applications in The Cloud.

Synopsis:

Cloud computing, with its promise of (almost) unlimited computation, storage, and bandwidth, is increasingly becoming the infrastructure of choice for many organizations. As cloud offerings mature, service-based applications need to dynamically recompose themselves to self-adapt to changing QoS requirements. In this paper, we present a decentralized mechanism for such self-adaptation, using market-based heuristics. We use a continuous double-auction to allow applications to decide which services to choose, among the many on offer. We view an application as a multi-agent system and the cloud as a marketplace where many such applications self-adapt. We show through a simulation study that our mechanism is effective for the individual application as well as from the collective perspective of all applications adapting at the same time.

6. PerLa: a Language and Middleware Architecture for Data Management and Integration in Pervasive Information Systems.

Synopsis:

A declarative SQL-like language and a middleware infrastructure are presented for collecting data from different nodes of a pervasive system. Data management is performed by hiding the complexity due to the large underlying heterogeneity of devices, which can span from passive RFID(s) to ad hoc sensor boards to portable computers. An important feature of the presented middleware is to make the integration of new device types in the system easy through the use of device self-description. Two case studies are described for PerLa usage, and a survey is made for comparing our approach with other projects in the area.