Researchers and practitioners alike agree that the waterfall approach to software development results in poor quality software systems. Unfortunately, the waterfall approach is inherently used in almost all of today’s development efforts resulting in system failures. The problem lies in the forward, linear development effort that produces inconsistent and incorrect specifications, designs, and code artifacts. It is proposed in this paper that a recursive software development process be used as a means of managing the complexity of today’s software systems. The recursive approach has the flexibility needed to perform development activities in any order to ensure that system requirements are met.

INTRODUCTION

Since the recognition that a “coding and debugging” process doesn’t produce defect-free software systems, software development process models have continued to evolve. The models range from a simplistic, sequential set of work activities as defined in the waterfall approach to more sophisticated cyclic work activities as provided by the incremental and spiral models. These software process models include a set of development phases that are typically performed in a predefined order based on top-down decomposition.
The waterfall software development process model was the first attempt at formalizing the development process by identifying an ordered set of work phases (Royce, 1970). The sequential nature of performing these phases, as shown in Figure 1, resulted in missed requirements, incorrect and incomplete designs, and various defects uncovered late in the development process, among other problems. As a result, the waterfall approach is now considered an unacceptable if not an obsolete approach to software development because of its missing feedback loops. However, it laid the groundwork for developing more rigorous process models for software development.

The incremental development process model supposedly addressed the weaknesses of the waterfall model by decomposing a system into increments each of which is completed using a set of development activities. Each increment, as shown in Figure 2, may be viewed as a fully functional “end-to-end” component that can be readily integrated into previous increments. A major drawback to this approach is that the decomposition of a system into increments may not be a straight-
Related Content

Security Testing Framework for Web Applications
[www.igi-global.com/article/security-testing-framework-for-web-applications/207728?camid=4v1a](www.igi-global.com/article/security-testing-framework-for-web-applications/207728?camid=4v1a)

Neural Network Control of a Laboratory Magnetic Levitator
[www.igi-global.com/chapter/neural-network-control-of-a-laboratory-magnetic-levitator/79673?camid=4v1a](www.igi-global.com/chapter/neural-network-control-of-a-laboratory-magnetic-levitator/79673?camid=4v1a)

A Rigorous Model for RAISE Specifications Reusability
[www.igi-global.com/chapter/rigorous-model-raise-specifications-reusability/28111?camid=4v1a](www.igi-global.com/chapter/rigorous-model-raise-specifications-reusability/28111?camid=4v1a)

A Framework for Intelligent Service Discovery
[www.igi-global.com/chapter/framework-intelligent-service-discovery/28120?camid=4v1a](www.igi-global.com/chapter/framework-intelligent-service-discovery/28120?camid=4v1a)