

Software Engineering Factory

by

Table of contents

1 Introduction.....	2
2 Motivation.....	2
3 Vision.....	2
4 References.....	3
4.1 Process.....	3
4.2 Requirements.....	3
4.3 Analysis & Design.....	3
4.4 Change Management.....	4
4.5 Configuration Management.....	4
4.6 Test Management.....	4
4.7 Build Management.....	4
4.8 Deployment Management.....	4
4.9 Software Development Asset Management.....	4
4.10 Continuous Integration.....	5
4.11 Model Driven Architecture.....	5
4.12 Service-Oriented Architecture.....	5
4.13 Autonomics.....	5

1. Introduction

The goal of the Software Engineering Factory (SEF) project is to provide a one-stop-shop software engineering environment to a development community. It is based on the following principles:

- Distributed Collaborative Development
- Continuous Integration
- Integrated Project, Change, Configuration and Test Management
- Model Driven Architecture
- Service Orientated Architecture, Software Development Asset Management and Reuse
- Autonomics

See project [Trac](#) Wikifor more information.

2. Motivation

Many tools exist to aid developers with software engineering. Some will claim that all you need is vi and a c compiler for write code. To them I say that if the code is delivered on time and on budget, is bug free and perfectly meets user requirements, then they are right! I am not one of those people. I need all the help I can get.

It gets even tougher when many people have to work together, and unlike computers, people tend to misunderstand each other, be late etc. all resulting in well publicised failed computer system deployments.

To overcome these problems, a number of methodologies have emerged. In turn, tools have evolved that feed off these methodologies, and feed back into them. They provide the glue which binds all the of people involved in the software engineering process together.

The combination of tools and services has resulted in some powerful solutions. As a result, developers now have to install an ever increasing number of tools on their workstation, and configure an ever more complex environment. Emergence of IDEs has helped, but not solved the problem. More and more companies are now providing hosted solutions e.g. SourceForge, CollabNet.

The hosted solutions provide powerful environments, however they are largely not a one-stop-shop i.e. some will provide bug tracking and version control, others will add project mangement facilities, some will deliver continious integration, and yet another group will provide application server environments. I think we can do better.

3. Vision

Software Engineering Factories should be implemented as an internally or externally hosted service. The service is a partnership of a Software Engineering Process (SEP) and Software Engineering Tools (SETs). The process for any project should be as simple as:

- Create new project.
- Capture and document requirements.
- Feed use-case, analysis, design, implementation and test models into SEF by adding them to the version control system.
- SEF will then build, test and deploy the derived executables to an application server.
- Developers and testers can now view the results.

4. References

4.1. Process

- Rational Unified Process (RUP)
 - IBM donate parts of RUP called [Basic Unified Process](#) (BUP) to [Eclipse Process Framework](#) project [proposal](#).
 - Meta-model will be based on [first draft](#) of [SPEM v2.0](#), next version of [Software Process Engineering Meta-model \(SPEM\) v1.1](#).
- eXtreme Programming (XP)
- DSDM
- PRINCE 2
- Personal Software Process (PSP) / Team Software Process (TSP)
- Capability Maturity Model for Software (SW-CMM)
- [PWAIN](#)

4.2. Requirements

- IBM Requisite Pro
- xmlbasedsrs

4.3. Analysis & Design

- ArgoUML
- IBM Rose
- TogetherJ
- [Catalog](#) of OMG Modeling and Metadata Specifications.

- Booch's [Handbook](#) of Software Architecture.

4.4. Change Management

- [IBM ClearQuest](#)
- Bugzilla
- JIRA
- [Trac](#)

4.5. Configuration Management

- [IBM ClearCase](#)
 - Workspace Versioning and Configuration Management ([JSR 147](#))
- Subversion
- CVS

4.6. Test Management

- [Mercury TestDirector](#)
- Mercury WinRunner
- Mercury QTP
- Mercury LoadRunner
- JUnit
- [STAF](#)
- Eclipse [Test & Performance Tools Platform](#) (TPTP) Project.

4.7. Build Management

- Apache ANT
- Apache Maven

4.8. Deployment Management

- IBM Tivoli

4.9. Software Development Asset Management

- LogicLibrary's [Logidex](#)
- [Koders](#)

- Reusable Asset Specification ([RAS](#)) v2.2
- [RAS for Workgroups](#).
- [Reuse engineering for SOA](#)
- [Mercury IT Governance Center](#)

4.10. Continuous Integration

- Read IBM [article](#) on Realising CI.
- Apache Gump
- CruiseControl
- [Bitten](#)

4.11. Model Driven Architecture

- [OMG's MDA](#)
- [AndroMDA](#)

4.12. Service-Oriented Architecture

- [Service-Oriented Architecture \(SOA\) and Web Services](#)
- [What is Service-Oriented Architecture?](#)
- [Web Services and Service-Oriented Architectures](#)

4.13. Autonomics

IBM is doing a lot of research in the area of [autonomics](#). The idea is simple - build computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies. This new computer paradigm means the design and implementation of computer systems, software, storage and support must exhibit these basic fundamentals from a user perspective:

- *Flexible*. The system will be able to sift data via a platform- and device-agnostic approach.
- *Accessible*. The nature of the autonomic system is that it is always on.
- *Transparent*. The system will perform its tasks and adapt to a user's needs without dragging the user into the intricacies of its workings.

Some useful links:

- [XML policy using Policy Management for Autonomic Computing](#)
- [New to Autonomics](#)

