System Variants

Rational Software White Paper
TP 155

Rational®
the software development company

Table of Contents

Introduction **1**

Variants of the System **1**

Different Parts of the System 1
Different Languages 1
Multiple Platforms 2
Patch Releases 2

Variants of Subsystems **3**

Mechanisms for Variability **4**

Effect on the Rational Unified Process **4**

Effect on the Implementation Workflow 4
Effect on Other Workflows 4

Introduction

This paper discusses what variants of systems are and how to manage them. You do not need to read this to understand the Rational Unified Process—on the contrary, it should be treated as an extension to the Rational Unified Process. The last section briefly discusses how the Rational Unified Process would be affected by the introduction of variants and variability.

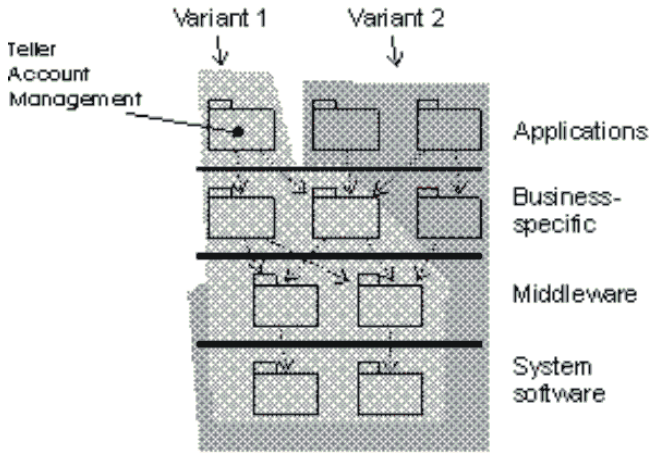
This is an area in which the Rational Unified Process will improve and expand in the future. This paper gives a first taste of that.

Variants of the System

Many systems are delivered in more than one variant. This means that the system is configured, packaged, and installed differently for different (classes of) customers. Sometimes, different variants are achieved simply by installing and adapting the system differently. Other times the variability is achieved by delivering different parts of the system to different customers. The following sections contain some examples of variants.

Different Parts of the System

Different parts of the complete system are delivered to different classes of customers. For example, a banking system is delivered as two different products. Let's say variant 1 of the system contains everything about telephone banking, whereas variant 2 contains everything about teller account management. The executables are defined in the subsystems in the applications layer. This means that a variant (variant 1 in the following figure) is a build with, for example, the subsystem Teller Account Management, and all the subsystems it needs to compile and execute—that is, all the subsystems it imports directly or indirectly.



A banking system developed as two variants.

Different Languages

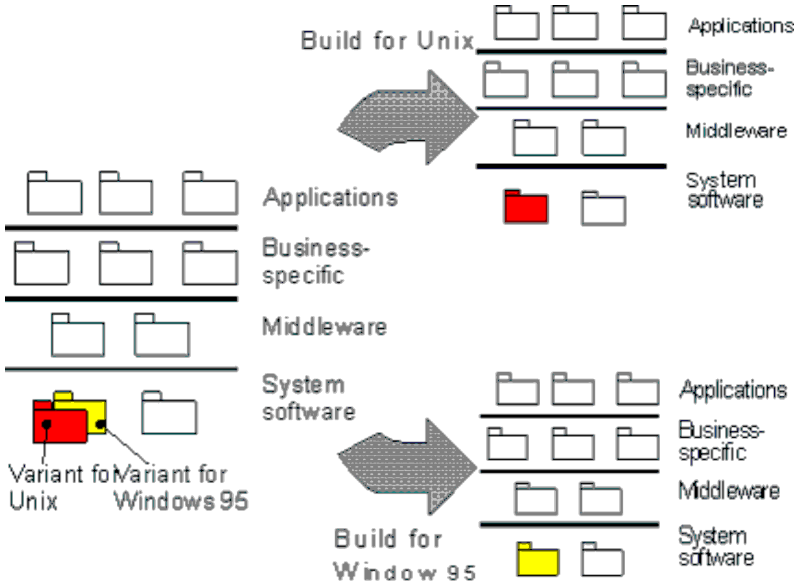
If the system is produced for different languages, for example, English, French, and Japanese, you want to deliver a variant of the system for each language. The difference between the variants is that all text, such as menus and help texts, should be in the specific language.

One way of handling different languages is to collect all texts in a file and have one file for each language. To deliver a system for a specific language means you deliver everything, including the file that contains the text for the specific language. This file is then read by the software at startup time, and all relevant variables are initialized.

Multiple Platforms

If the system supports multiple, incompatible platforms, then one variant of the system is needed for each platform. For example, if a system runs on both Windows NT and UNIX, then two variants of the system are produced.

In the example that follows, the platform-specific code is located in one subsystem. In this case, two variants of subsystem are developed. A compilation file—a "makefile"—specifies which version of each source code file should be compiled together. You could also say that a makefile specifies which variant of each subsystem should be part of the build. Therefore, you need one makefile for each platform.



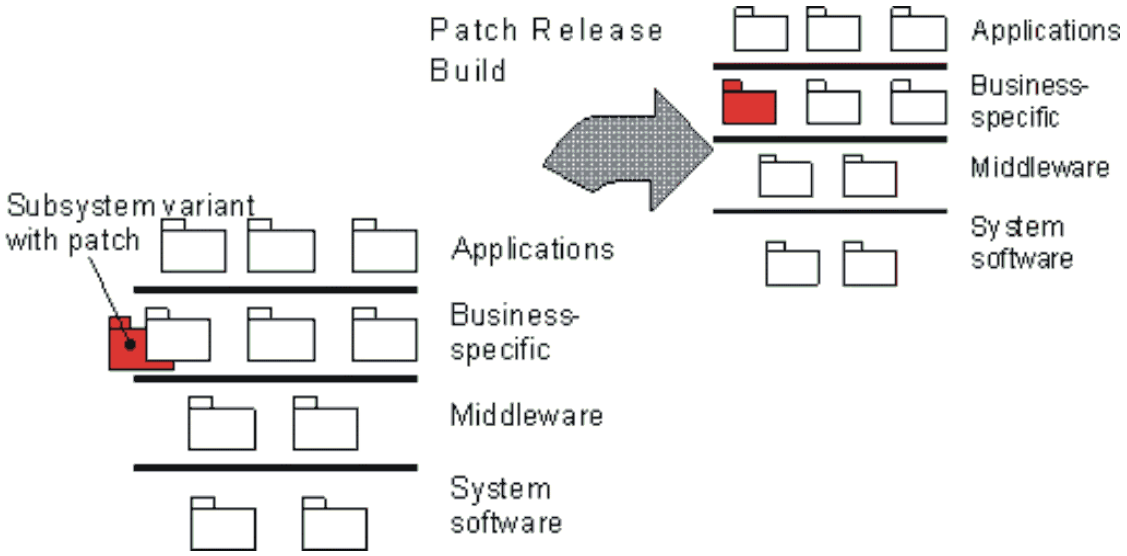
A system is developed to run on several platforms.

Patch Releases

At times it's necessary to develop a patch release of the system. This is normally done in parallel with a development project, which means that the patch release is another variant of the system; it exists in parallel with the "main" version of the system.

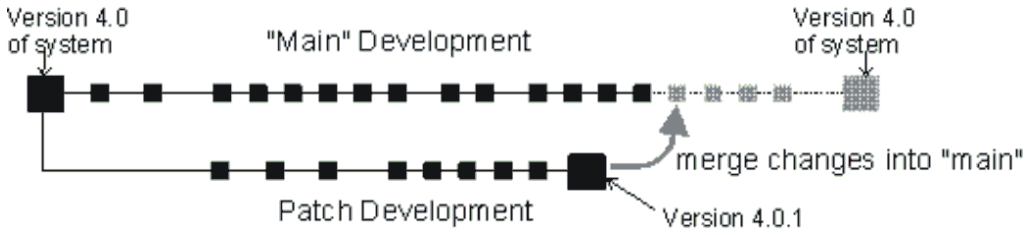
The development of a patch release is done in parallel with the main development project. Gray squares indicate future releases and baselines.

The changes you need to make are located in one or several implementation subsystems. Because a normal development project is going on in parallel, you need to develop variants of these subsystems in parallel with the main development effort. To create a build, you specify in a makefile which variant of each subsystem should be part of the build.



A patch-release build is created with a variant of the subsystem that contains the patch.

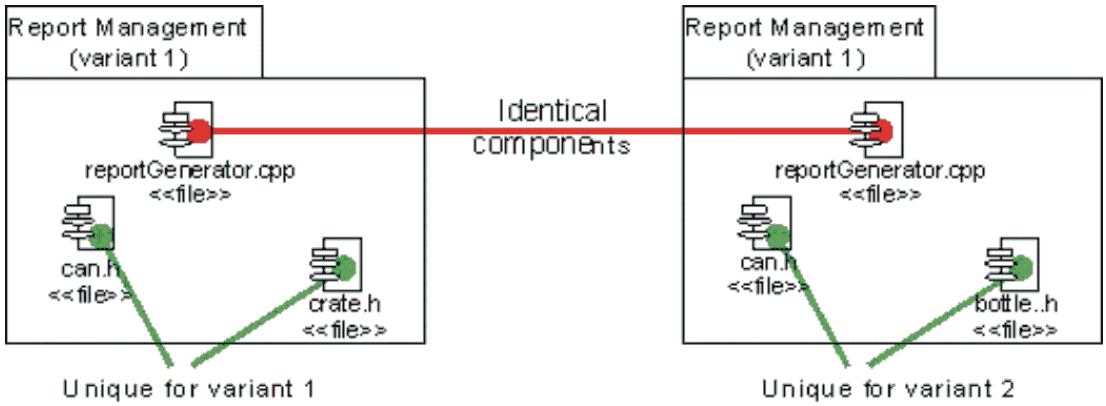
This kind of variant normally has a short lifespan. After the patch is released, this variant is not developed any further and all code changes of value are incorporated into the main development path.



The relevant changes in the patch release are merged into the main development path when it's convenient.

Variants of Subsystems

The previous examples illustrate that you often need two or more variants of one specific subsystem. These variants will have some components in common, whereas other components are unique for each subsystem variant.



Two variants of an implementation subsystem called Report Management where one component is decided to be the same in both variants. The other components are unique for each variant.

Frequently, each variant of a subsystem is developed by one implementer and the subsystem's variants are developed in parallel. If the component is changed by one implementer developing one variant, then the change should be propagated to the other variant. To make this possible, you need tool support from a **configuration management and version control tool** (CMVC) to manage the components that should be identical in the two variants.

Mechanisms for Variability

There are several mechanisms to create system variants. Some of them have been mentioned in the previous sections. Each mechanism has different characteristics. Normally, you use a combination of these mechanisms to create variability in your system.

- **Compilation (and link) files**. Specify in a compilation file—a makefile in a Unix environment—which variant of each source code file should be compiled and linked together into executables.
- **Dynamically loaded components**. Develop parts of the system as dynamic link libraries, applets or Active X components that can be linked into the running program at runtime. These components can then be managed by a CMVC tool, which makes it possible to deliver subsets of these to a customer.
- **Startup files**. Use files that contain information the software reads when the system is started to initialize the system. For example, resource files in Windows, startup files or initialization files, which are read by the software when the system is started, and sets up the system differently. Use this, for example, if the system is to be customized for different languages, in which case you keep all texts in text files that are read when the system is started.
- **Divide the system in several executables**. Develop the system as several executables; for example, .exe files. Combinations of these can be delivered to the customer. The various combinations of executables are managed by a CMVC tool.

Effect on the Rational Unified Process

This section briefly describes how the Rational Unified Process would be affected by the introduction of variants.

Effect on the Implementation Workflow

The Implementation workflow needs to be extended in the following areas:

- Add the following steps to the activity **Define the Implementation View**:
  - How to define which variants of the system should be developed
  - How to decide which subsystems should have variants
  - How to decide what variability mechanism to use to achieve variability
- In the activity **Plan System Integration**, you must take variants into consideration and plan how to integrate different system variants.
- You need to describe more details about parallel development "between" variants of a subsystem. For example, what happens if an "identical" component should be severed, or how are components merged? This also affects several of the implementer's activities.

Other implementation activities may also be affected.

Effect on Other Workflows

Developing variants, or families of systems, affects all workflows. In the previous sections, some effects on the Implementation workflow have been discussed. The following is a brief list of how the other workflows will be affected:

- **Requirements Capture**—should describe how you identify the variants of the system. What each class of customer wants.

- **Analysis & Design**—should describe how you model variants in the design model, how you design subsystems variants, and how you define variants.
- **Test**—should describe how to test a family of systems (variants). Test each variant of the system as a separate system. Variants also impact integration test.
- **Management**—you need to organize the project, maybe in separate teams for each subsystem variant. In a large project, you may even have separate project managers for each system variant.
- **Environment**—you need tools that help manage systems variants.
- **Deployment**—is much more complex because there are several classes of customers to which you'll deliver your final product.

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

For International Offices: www.rational.com/worldwide