



## White Paper

---

### Efficient Management of Tests and Defects in Variant-Rich Systems with pure::variants and IBM Rational ClearQuest

#### **Publisher**

© pure-systems GmbH  
Agnetenstrasse 14  
39106 Magdeburg  
<http://www.pure-systems.com>

#### **Contacts**

**Dave Gööck**  
pure-systems GmbH  
[dave.goeoeck@pure-systems.com](mailto:dave.goeoeck@pure-systems.com)  
+49 391 544569 41

**Taha Boulaguigue**  
IBM Deutschland GmbH  
[Boulaguigue@de.ibm.com](mailto:Boulaguigue@de.ibm.com)

## Table of Contents

1.	Introduction .....	3
1.1.	Variant Management in the Product / Application Life Cycle .....	3
1.2.	Variant Management in Test Management & Defect Tracking.....	4
1.2.1.	The use for Deployment, QA, Analysis, Visualization .....	4
1.2.2.	Improvement of Test Management / Defect Tracking .....	4
2.	Introducing the Solution.....	5
2.1.	IBM Rational Tool Environment.....	5
2.1.1.	IBM Rational Software Delivery Platform.....	5
2.1.2.	IBM Rational ClearQuest .....	5
2.2.	Variant Management with pure::variants .....	7
2.2.1.	Technology Fundamentals.....	8
2.3.	pure::variants Connector for ClearQuest .....	9
3.	Test management example:.....	11
3.1.	Preparation of ClearQuest .....	11
3.2.	Preparation of pure::variants .....	11
3.3.	Create and configure tests in ClearQuest.....	11
3.4.	Link tests with pure::variants .....	12
3.5.	Create test logs in ClearQuest .....	13
3.6.	Show test results in pure::variants.....	13
4.	Defect Tracking example: .....	15
4.1.	Preparations.....	15
4.2.	Link defects with pure::variants.....	15
4.3.	Show defect status information in pure::variants .....	16
4.4.	Change the defect state in ClearQuest.....	17
5.	Future development of the Connector .....	18
6.	References.....	19

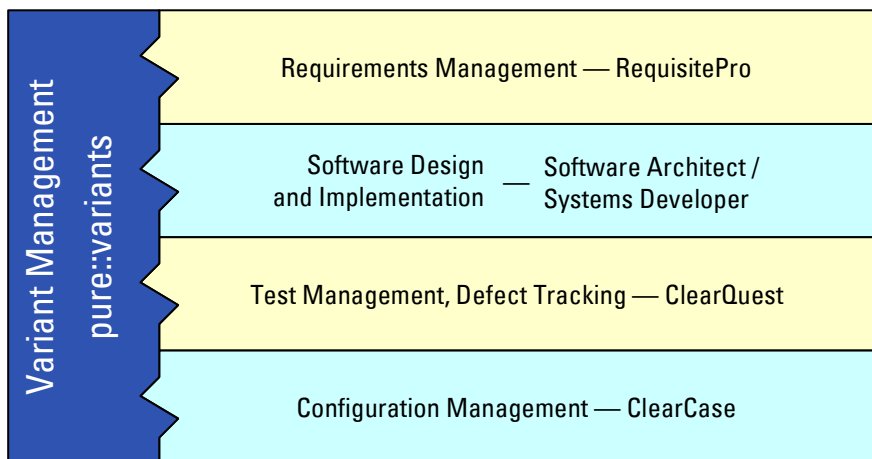
## 1. Introduction

Related products frequently share much of the same software, with only a few differences realizing product-specific functionality. The product line approach separates parts of products into commonalities and differences. Commonalities being those components that are contained in every product variant. Efficient use of this approach is a challenge but in its implementation – variant management – there is high potential to significantly increase the efficient use and reuse of resources. With variant management, knowledge about the functionality of the different product features, requirements and dependencies will be managed much more efficiently, kept more accessible and can consequently be utilized more successfully.

This white paper shows how to manage defects and tests in systems with many variants in a more efficient way, by the use of variant management. This approach can have huge potential, particularly with regard to software product lines.

### 1.1. Variant Management in the Product / Application Life Cycle

Variant management is a general activity, which plays an important role in the entire life cycle of product (application) development. For this reason a smooth and effective integration of a variant management tool into the constituted tool environment is necessary for maximum efficiency.



Variant Management and the Application Life Cycle

A binding to an existing requirements management tool can be used for automatic generation of feature models (see section 2.2.1.1 Feature Modelling), to enrich them with variability information and to make requirements information available in other steps of the development process. By linking variant management with the tools used for design and implementation purposes, simple import of family models (see section 2.2.1.2 Family Modelling) becomes possible and therefore efficient development of variant specific implementations. Beyond that if configuration management is enriched to handle variants, generation and building of specific solutions can be done easily.

But the product life cycle is not finished with the finished implementation of software, the domains of test management and defect tracking become more important, particularly with the growth of software complexity, so it is indispensable to devote particular attention to the build of variants. Furthermore, the availability of information about the workflow-state of the different components of the product line with respect to their variability or variant membership can increase the product quality as well as the effectiveness of the development process.

## ***1.2. Variant Management in Test Management & Defect Tracking***

As described, variant management is a general activity, which must also be dealt with in defect tracking and test management. Often not all defects and tests are relevant for all product variants. This knowledge must be acquired and used, in order to be able to avoid unnecessary development and testing effort. It is important in principle to be able to realize complete variant-specific traceability. Thus expenditure and risk for new problems can be measured more accurately. For example, problems in delivered product variants can be treated differently from errors or missed tests in undelivered variants.

There are two important approaches that must receive more attention.

### **1.2.1. The use for Deployment, QA, Analysis, Visualization**

On the one hand the evaluation of variant-related data like test results and status information is a big challenge. For example, test management tools typically offer efficient ways of structuring test cases logically, of administering test logs efficiently and for allowing extensive evaluations, however it does not represent information about product variants and restrictions or constraints of the product line. Since modelling variants by their features and dependencies is usually hard to implement or very unpractical in test management, a link with a tool specifically designed for variant management can combine the efficiencies of both platforms in the best possible way. For this purpose test cases can be linked with individual elements from variant management and thus be included in modelling, so that the relevant status information is present for variant-specific analysis and visualization. In addition the information is available for further use in the workflow of product development, so that it can be used for deployment processes, extensive quality management and within other domains.

### **1.2.2. Improvement of Test Management / Defect Tracking**

On the other hand the information provided by variant management gives the potential to obtain an increase in efficiency particularly of test management but also of defect tracking. The following scenarios can be imagined:

- Automatic generation of variant-specific test suites, in order to accelerate and guarantee the processes of preparation and testing. Thus all **relevant** tests can be considered for the test run of a variant.
- Automatic generation of "stubs" for test plans, test cases, etc. for instance to guarantee a desired minimum test coverage for all variants of the product line or to accelerate the test management process.
- Automatic generation of variant-specific defect lists, which contain exactly the defects, which concern the actual variant.

## 2. Introducing the Solution

In the following a solution is presented, that uses IBM Rational ClearQuest for test management and defect tracking and pure::variants from pure-systems for variant management.

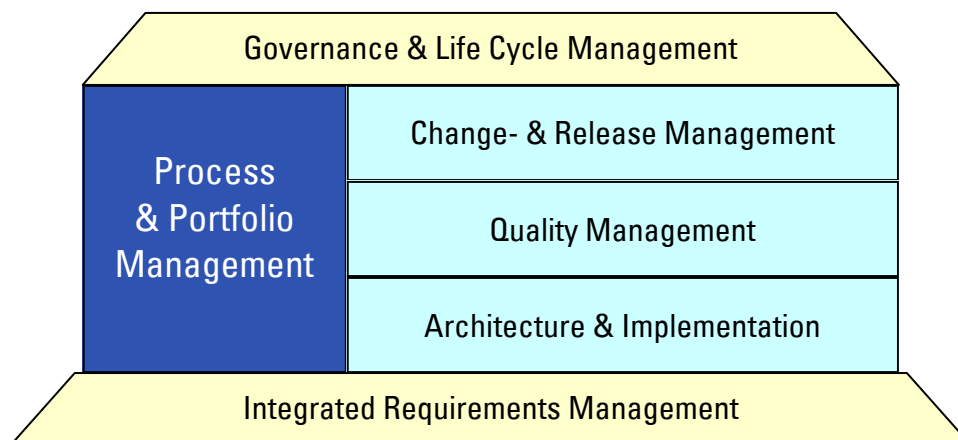
### 2.1. IBM Rational Tool Environment

ClearQuest is developed and distributed as part of the IBM Rational Software Delivery Platform. It is used primarily for test management, defect tracking and change management.

#### 2.1.1. IBM Rational Software Delivery Platform

The IBM Rational Software Delivery Platform is a complete and general development platform, which covers the entire life cycle of applications. With this platform all kinds of software solutions can be developed and used for all industries. For example applications for web, client/server, desktop, real time, embedded systems, etc.

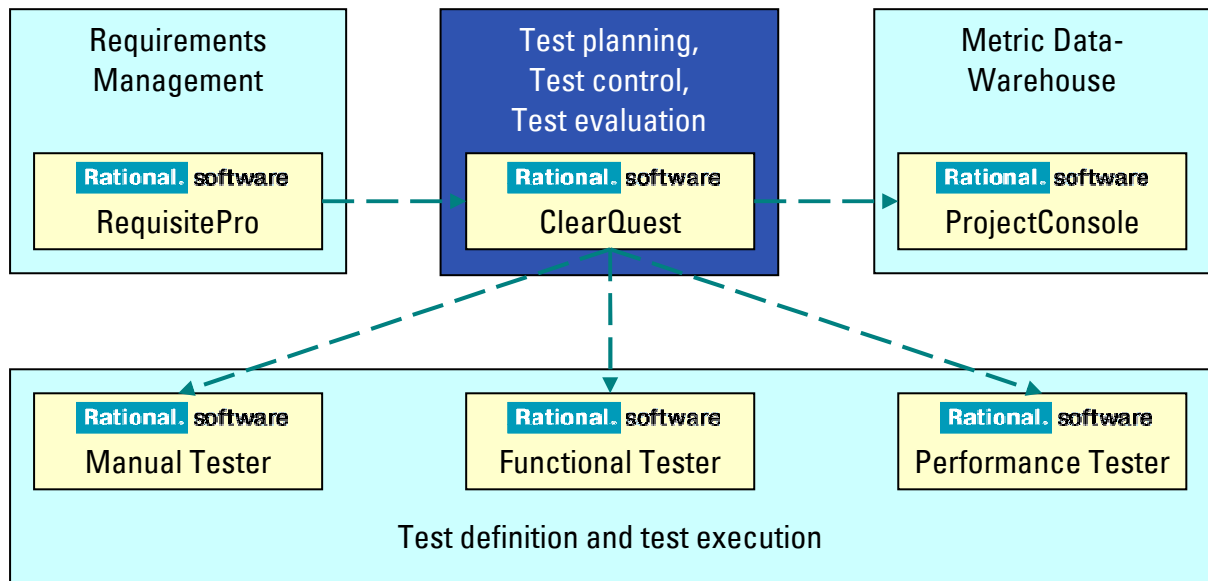
The IBM Rational Software Delivery Platform is based on a platform, which makes unparalleled automation and teamwork possible during the entire development process: the universal development framework Eclipse. Eclipse is an open source software development project with over 100 software vendors as active participants. Eclipse facilitates role-based user interfaces, which offer an adapted view of the general enterprise and development resources to each member of the team. With Eclipse as central platform, the IBM Rational Software Delivery Platform is supported by a multiplicity of partners and extended by a dynamic ecosystem of tools and services.



IBM Rational Software Delivery Platform

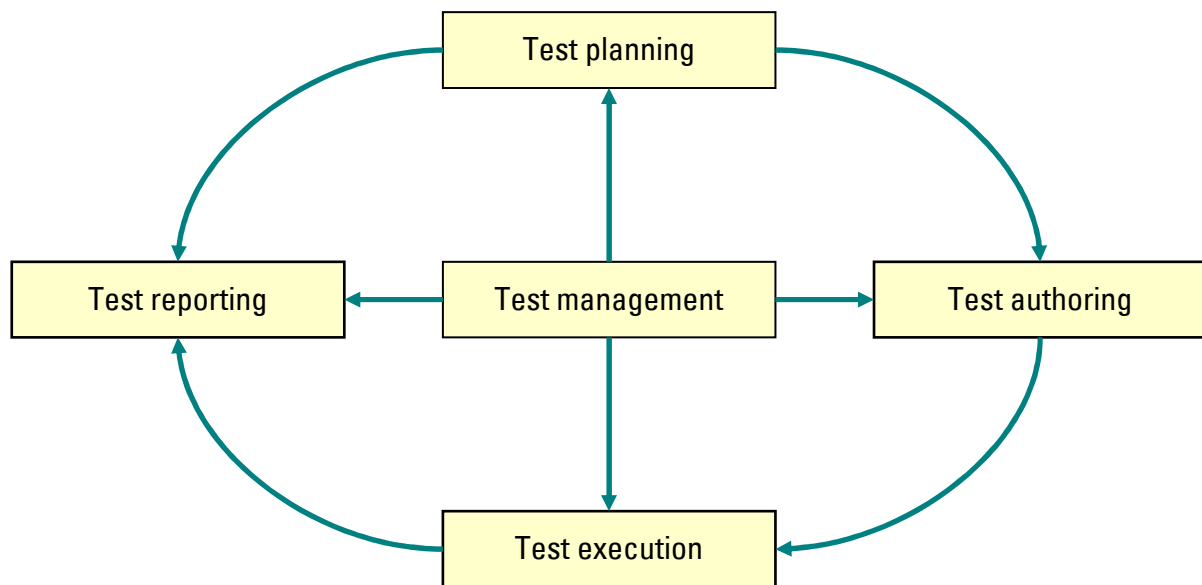
#### 2.1.2. IBM Rational ClearQuest

In order to guarantee the quality of a software product at all levels, IBM Rational offers a wide range of Software Quality Management Tools that address all dimensions of software quality including functionality, reliability, security, compliance and performance.



Overview of the IBM Rational Tools for the Tester

IBM Rational ClearQuest is a tool for enterprise test life cycle management, with integrated defect and change management.



IBM Rational ClearQuest Test Management Model

The Test Management Model consists of the following areas:

- **Test planning**
  - Test case creation
  - Test case lifecycle management
  - Test assets organization
- **Test authoring:** creating test scripts for test case execution

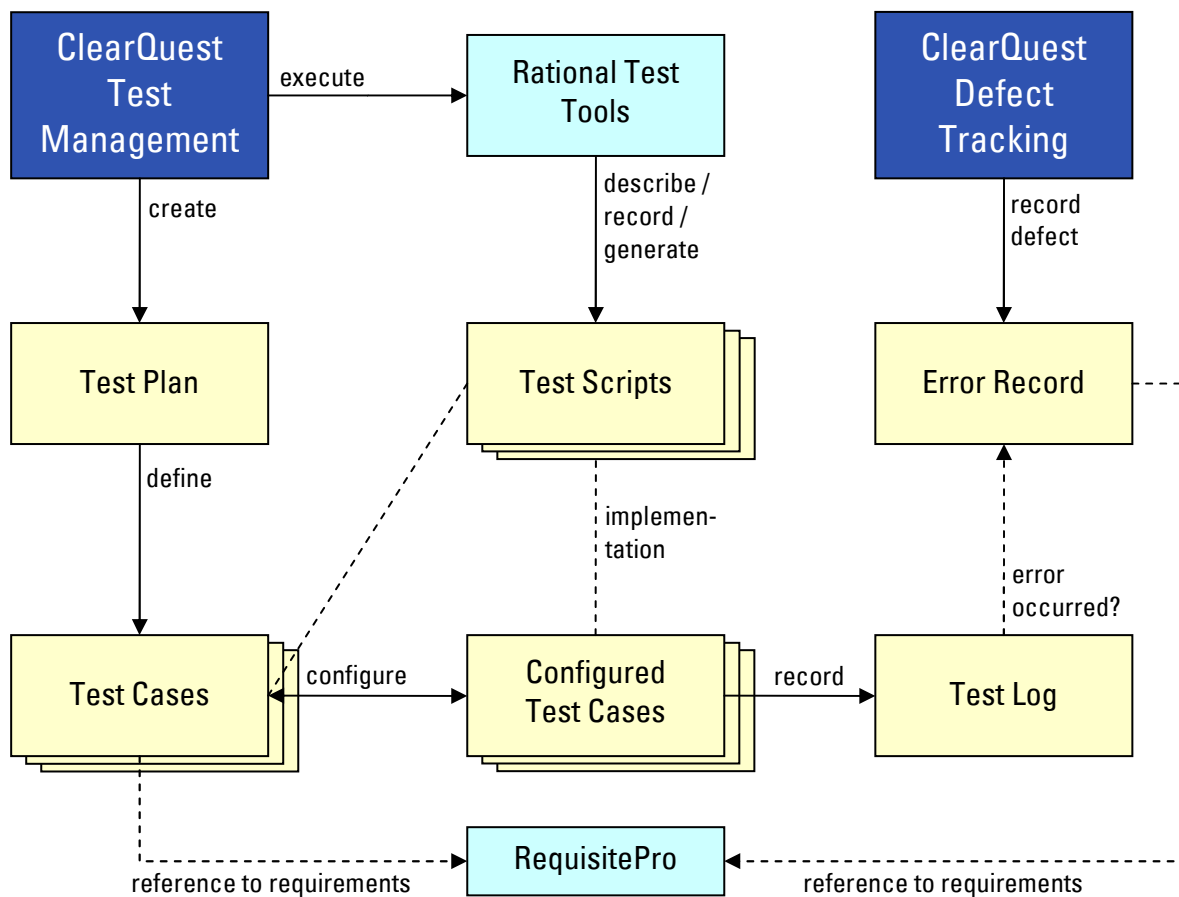
- **Test execution:** running configured test cases or test suites, and committing the results to the database
- **Test reporting:** user-defined charts, queries, and reports

These stages are sequential. Typically, a Test Lead sets up the necessary items in the planning phase. A Tester writes test scripts in the Authoring phase and then runs tests to gather test results in the Execution phase. Throughout all three phases, ClearQuest reporting can be used to monitor progress.

Test Plans in Rational ClearQuest are a kind of folders where related test cases can be defined. Through the allocation of further characteristics such as test iteration, configuration and a test script (manual or automated) the test cases becomes executable units: configured test cases.

Single configured test cases or test suites are executed using the selected testing tools and the test results are logged in Rational ClearQuest. Defects and deviations can then be handed over directly into defect tracking.

The following diagram shows the corresponding workflow:



ClearQuest Test Workflow

## 2.2. Variant Management with pure::variants

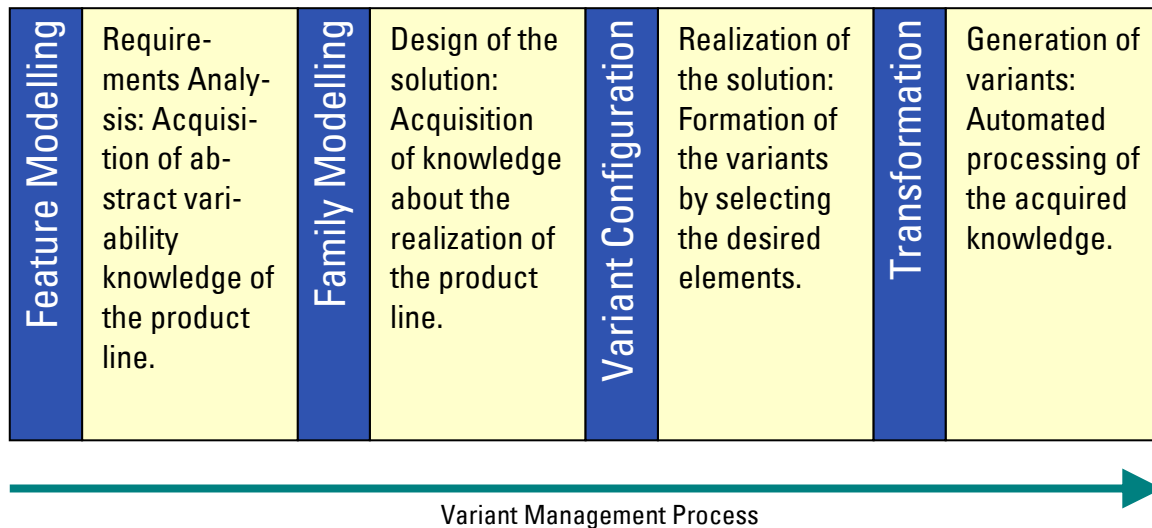
pure::variants provides all characteristics necessary for a successful, tool-supported development and realization of product lines. It is applicable throughout all phases of development, starting from requirement analysis and software design, through implementation and on to

production, testing, use and maintenance. All basic mechanisms are independent of any concrete programming language, development technology or platform.

pure::variants also has outstanding integration abilities. It can be integrated easily into existing development processes and environments. With pure::variants a uniform solution results from the integration of disparate customer tools for configuration management, requirement engineering, modelling and testing. Binding to arbitrary tools is possible at any time using an XML-based exchange format and a multiplicity of open interfaces. Efficient import mechanisms make integration into existing software projects possible and reduce the initial expenditure substantially. For highly-specific requirements, pure::variants is flexibly expandable with the integration of additional modules. All modules have complete access to all variant knowledge.

## 2.2.1. Technology Fundamentals

With pure::variants the process of variant management is divided into four parts/steps. Each part is addressed at different stages in the development life cycle.



### 2.2.1.1. Feature Modelling

Based on a textual analysis of the requirements, feature models are created, the features of the product line and the dependencies between each other are the focus of modelling. Feature models can be structured hierarchically, displayed graphically and are easily understandable.

With connections to other tools used in the product line development life cycle, the raw data for feature models can often be extracted from an existing requirements database and then augmented with variant-related information. Furthermore it is common that there exists some additional information associated with each feature of a product line; for example, test cases and their current execution status, change requests, defects, workflow information, and so on. This information is usually collected in other tools and databases and so is available for further processing, and can be linked with features in pure::variants.

### 2.2.1.2. Family Modelling

Based on the feature model and further requirements for the concrete derivation of the software systems and products, the design of the solution(s) is accomplished. Solution elements with their relations, restrictions and links to their requirements are gathered together in the family model and are thereby available for automatic processing.



It is also possible to generate the raw data for the family models out of the implementation of the product line and to augment this raw data with relevant variability information. In this case a connection to an existing configuration management tool is often an effective solution. In addition to modelling the solution, solution components are linked with features in the feature models. Thus a continuous consideration of variability from the requirement to the solution is possible. As for the features of the product line a multiplicity of additional information can also arise during its implementation, which correspondingly can be set in relation to the elements of the family models.

### **2.2.1.3. Variant Configuration**

After all relevant knowledge about the product line and its solution have been acquired, individual products/variants can be derived. Through the connection of solution elements with the corresponding requirements, a selection of the desired features is sufficient to describe a specific solution and therefore to derive a variant. In addition the (now available) modelling of dependencies and restrictions makes it possible to check the desired configuration for validity, and so pure::variants is able to support the user while configuring a variant correctly.

Any additional information linked to model elements can be processed, so that metrics can be provided easily, in order to gain variant-specific information about the state of implementation, test coverage, defect ratio and so on.

### **2.2.1.4. Transformation**

The configuration of a variant provides a description of its solution, which may be used to automatically generate the final product variant. For this purpose pure::variants provides a standard transformation. This transformation can be customized extensively and can be extended with additional (customer) modules, in order to control the transformation process.

Since modelling information from all different stages of the development life cycle was linked together, all this information is now available during the transformation process. Depending on which transformation modules have been integrated into pure::variants, different actions are now performed during the transformation. For example, a build process for variant generation can be started, for which a variant-specific composition of sources and a makefile were previously generated.

Test management information can be used to generate variant-specific test suites. A link with defect tracking makes the generation of variant-specific error lists a further possibility.

## ***2.3. pure::variants Connector for ClearQuest***

pure::variants Connector for ClearQuest offers a connection between variant management and test management/defect tracking. In section 1.2 the future potential of linking pure::variants and ClearQuest was discussed. Currently the focus is on visualizing the status of an element.

Defects and test cases recorded in ClearQuest can be linked to specific elements in pure::variants. This makes it possible to retrieve the last state of each ClearQuest item. In case of an error like a not yet fixed defect or a failed test, pure::variants reports an appropriate warning. Connections are made by means of the ID of the respective entry of the ClearQuest database and stored in a special attribute of the corresponding element in pure::variants. Link restrictions for attribute values can be added as usual to make them relevant only in some specific variants. Thus errors can be modelled that only occur in a concrete configuration of

**certain** components of the product line or tests that may only be executed and whose result is only relevant when **all** tested features are contained in the product.

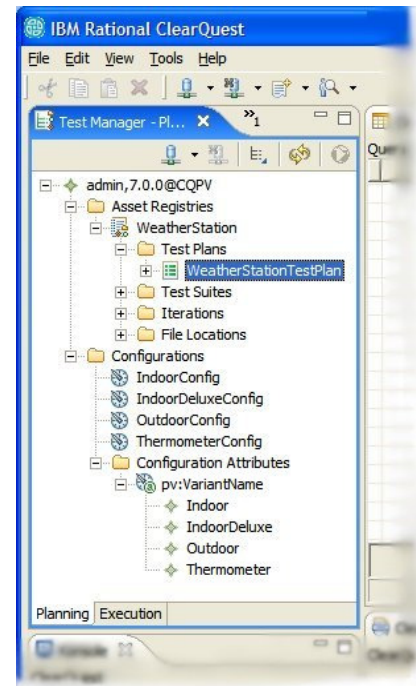
pure::variants Connector for ClearQuest keeps the connection to the ClearQuest database open and updates its status information at regular intervals. Therefore changes are directly visible in pure::variants and the information may be used to support the deployment process.

How pure::variants Connector for ClearQuest works is illustrated in the following examples – one a test management scenario, and one a defect-tracking scenario. Finally we give a brief outline of planned future development of the Connector.

### 3. Test management example:

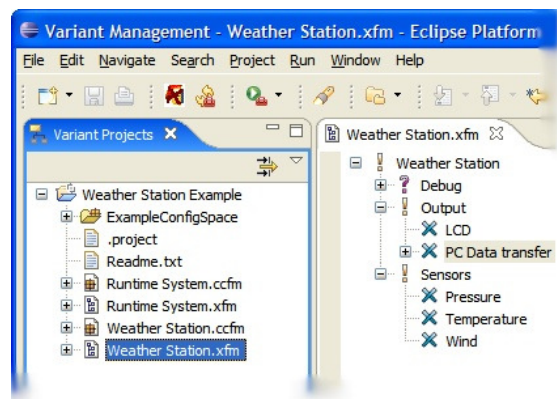
#### 3.1. Preparation of ClearQuest

1. Set up a ClearQuest user database *CQPV* in the ClearQuest Designer
2. Create a connection to the new database with the ClearQuest Eclipse Client
3. Add a new asset registry *WeatherStation* to the ClearQuest TestManager
4. Create a test plan *WeatherStationTestPlan*
5. Create a new configuration attribute *ps:VariantName* with one value for each desired variant
6. Create one configuration for each variant



#### 3.2. Preparation of pure::variants

1. Create the example project for the weather station (File->New->Example...->Weather Station)
2. Open the feature model *Weather Station.xfm* of the weather station project

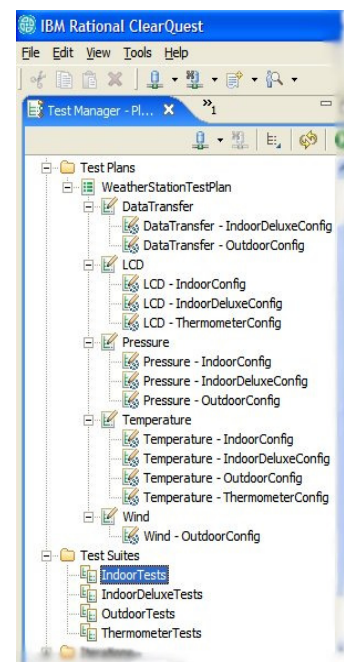


#### 3.3. Create and configure tests in ClearQuest

One possible representation of variants in ClearQuest is to use configurations, whereby in section 3.1 for each relevant product variant exactly one configuration was created. Test cases that are managed in test plans are independent of any concrete variant. This means that the unconfigured test cases cover the functionality of the entire product line.

The information about which variants a test case is associated with is represented by its configuration. Since there is exactly one configuration for each variant a configured test is created for each test that contains the tested functionality.

In the last step test suites for each variant can be created and configured. This makes it possible to test each variant exactly according to its specific functionality and implementation. In a

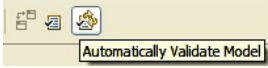


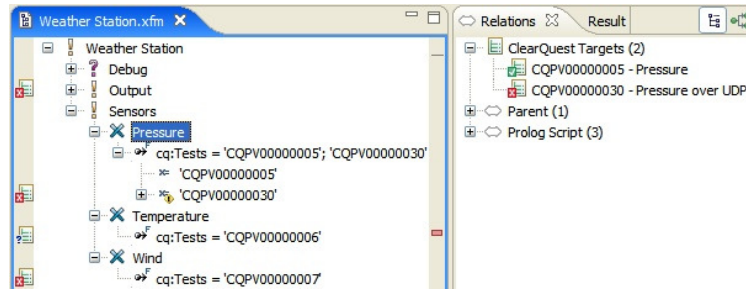
production scenario many steps can be initiated automatically using pure::variants.

1. Create test cases for the desired features from the weather station in the ClearQuest Eclipse Client. (e.g.: LCD, DataTransfer, Pressure, Temperature, Wind)
2. Create the desired configured test cases for each test case. For each variant where the tested feature is contained, a configured test is created.
3. Create and configure a test suite for each variant. All tests for a configuration will be assigned to the test suite.

### 3.4. Link tests with pure::variants

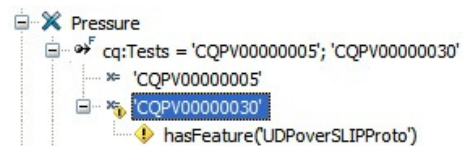
In order to get status information for specific variants from ClearQuest, the created test cases need to be linked with elements in pure::variants. Configured test cases are assigned to the corresponding variants automatically, by means of a new configuration attribute *pv:VariantName*.

1. Check the connector's settings in the pure::variants preferences.
  - a. In "Variant Management -> External Tools -> ClearQuest Tests", the *Local ClearQuest Test Settings* have to be set to the connection created in section 3.1.
  - b. In "Variant Management -> External Tools", *ClearQuest Tests* should be activated. To begin with it could be useful to set the *Update time* to a small value e.g. 10 seconds.
  - c. In "Variant Management -> Visualisation", *Show restrictions* and *Show attributes* should be activated.
  - d. In the pure::variants toolbar, *Automatically Validate Model*  should be turned on.
2. Select an element in the feature model and create a new link to a test by means of the context menu (New -> ClearQuest Test).
  - a. The connector uses a connection which was previously configured in the ClearQuest Client for communication with the ClearQuest database. When first connecting the Connector asks for the necessary authentication data. (In an empty ClearQuest database a user *admin* with no password is usually created.)
  - b. A dialog opens for test selection. All available test cases are listed. After selection of the desired test and completion of the dialog, an attribute *cq:Tests* has been added to the element with the test ID as value.



Tests in the editor and their visualization in the Relations View

- c. This process must be repeated for each test case which is to be linked with variant management in order to be available for further processing there. For each test that is linked to the currently selected element the *Relations View* shows a relation with status information fetched from ClearQuest.
3. In some cases conditional tests exist. These could test the interaction between different functionalities. Restrictions can be added to limit the validity of the respective test using the context menu of the respective attribute value for a test case.



### 3.5. Create test logs in ClearQuest

Create some test logs in ClearQuest to record the execution results of the tests in pure::variants.


1. Navigate to a configured test case in the Test Manager of the ClearQuest Eclipse Client.
2. Create a new test log by means of the context menu of the configured test case *New (Test Log)*. Complete the following dialog adding some short execution details and close the dialog.

To have more data for use in pure::variants it is useful to define test logs for several tests and set some to *passed* and others to *failed*.

### 3.6. Show test results in pure::variants

Now all information is available for evaluation and visualization in pure::variants. Test results are shown in pure::variants as follows:


1. The test failed:

In this case a marker  will be created at the left hand side of the model in pure::variants. It marks the element that links to the failed test. Thus where errors could occur in a variant is directly shown in the models.


In variant models the problem marker is shown only if the corresponding element is contained in the variant and the corresponding test was not removed from the solution by restrictions. In addition pure::variants tries to figure out which configured test case belongs to the corresponding variant by means of the configuration attribute *pv:VariantName* or the name of the configuration, so that only the relevant test logs are considered.

Note: The variant model must be checked to make the markers show. Either perform a single check by pressing the check button  or activate automatic checking. 

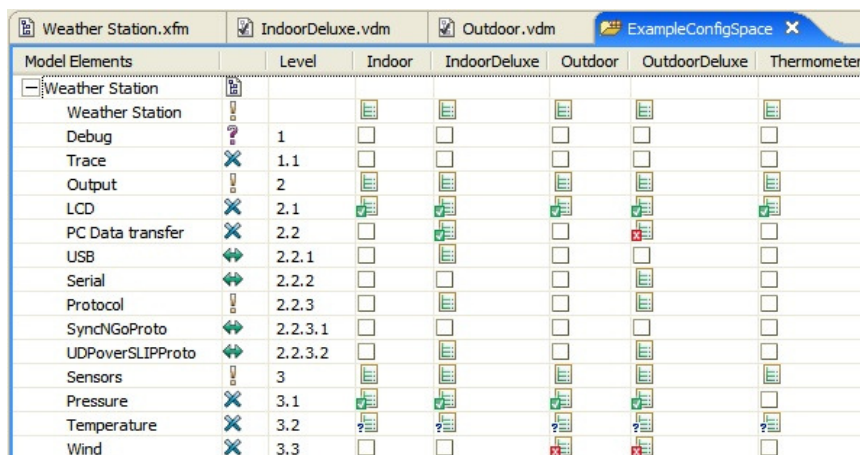
2. The test passed:

















































































In this case no markers are shown in the models. Only in the *Relations View* does the symbol  show that the test passed.

3. The test has not run yet or is in an unknown state:

In this case a marker with a question mark is generated . This makes it very easy to see that there is still need for action here and there was insufficient testing.

In addition to display of markers in the models, status information is shown in the Matrix View of a Configuration Space. Double-clicking on the Config Space opens an editor showing one column for each variant. Each row corresponds to an element and the selection status for each element is immediately apparent.



Model Elements	Level	Indoor	IndoorDeluxe	Outdoor	OutdoorDeluxe	Thermometer
Weather Station						
Weather Station						
Debug	1					
Trace	1.1					
Output	2					
LCD	2.1					
PC Data transfer	2.2					
USB	2.2.1					
Serial	2.2.2					
Protocol	2.2.3					
SyncNGoProto	2.2.3.1					
UDPOverSLIPProto	2.2.3.2					
Sensors	3					
Pressure	3.1					
Temperature	3.2					
Wind	3.3					

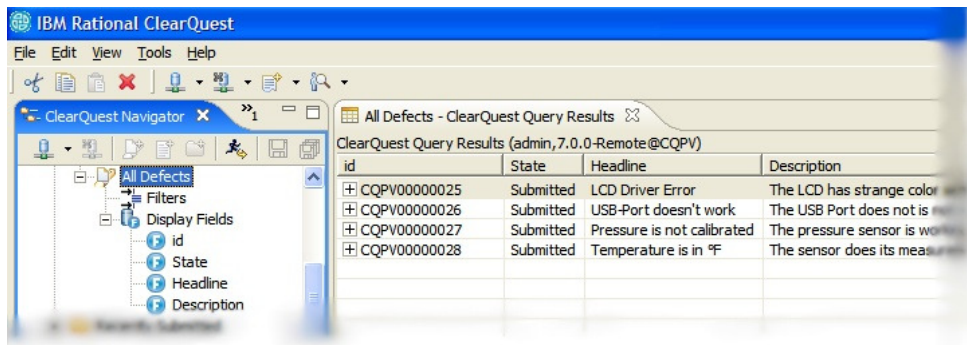
Variant overview – View of the marker symbols in the Matrix View

By selecting *“Visualize... -> ClearQuest Test State”* in the context menu of the Matrix View another view on the selections for the test states of the elements can be shown. With this it is evident in which variants failing tests occur or which variants must be still tested. The symbols here are the same as in the models.

## 4. Defect Tracking example:

### 4.1. Preparations

1. The preparations for setting up the ClearQuest database correspond to those in section 3.1. Furthermore it is useful to create a query to list all defects. The easiest way to do this is by means of the context menu of *Public Queries* in the *ClearQuest Navigator*, by creating a query on the defects without filters.
2. For variant management with defects, unlike the test cases example, it is usually unnecessary to devote special attention to their structuring. For this reason it is sufficient to create some defects in ClearQuest as a basis for this example. (E.g.: LCD, USB port, Pressure, Temperature)



Query to show all defects in ClearQuest

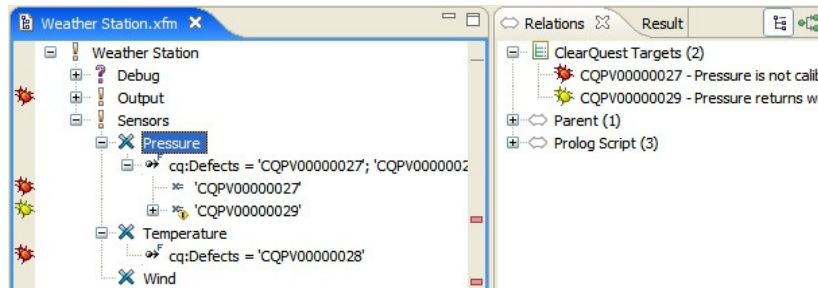
3. As described in section 3.2 the weather station example should be created in pure::variants.
4. The pure::variants preferences for defect tracking must be set in "Variant Management -> External Tools -> ClearQuest Defects" and in "Variant Management -> External Tools" as mentioned in section 3.4, part 1.

### 4.2. Link defects with pure::variants

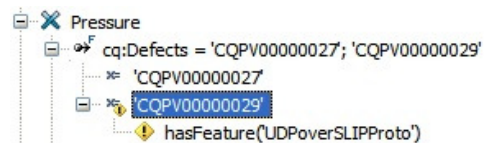
As done above for test management, defects in ClearQuest have to be linked with elements in pure::variants.

1. Select an element in the feature model and create a new link to a defect by means of its context menu (New -> ClearQuest Defect).
  - a. The Connector uses a connection that was previously configured in the ClearQuest Client to communicate with the ClearQuest database. When the first connection is attempted the Connector asks for the necessary authentication data. (In an empty ClearQuest database a user *admin* with no password is commonly created.)
  - b. A dialog for defect selection opens. All available defects are listed. After selecting the desired defect and completing the dialog, an attribute *cq:Defects* is added to the element with the defect ID as its value.
  - c. This process is followed for each defect that is to be linked with variant management in order to be available for further processing there. For each defect

linked to the currently selected element the *Relations View* shows a relation with status information fetched from the ClearQuest database.





- By using the context menu of the respective attribute value for a defect it is possible to add restrictions to be able to evaluate the current processing state of a variant to add certain conditions to a defect (E.g.: An error, which arises only in interaction with other functionality and is therefore not relevant in each variant.)



### 4.3. Show defect status information in pure::variants

After these steps all information is now available for evaluation and visualization in pure::variants. The state of a defect is shown in the user interface of pure::variants as follows:


- The defect is open/not yet assigned:

In this case a marker will be created at the left hand side of the model in pure::variants. It marks the element that links to the open defect. So, where errors occurred/may possibly occur in a variant is directly visible. If the defect was classified as *Enhancement* or *Minor*, its symbol is yellow , at a higher severity the marker is red .


The problem marker is shown in the variant models only if the corresponding element is contained in the variant and the associated defect was not removed from the solution by restrictions.

Note: The variant model must be checked to make the markers show. Either perform a single check by pressing the check-button  or by activating automatic checking .

- The defect is closed:

In this case no markers are shown in the models. Only in the *Relations View* does an appropriate symbol  show that the defect is closed.

- The defect is in an unknown state:

Here a marker with a question mark is generated . This happens rarely and particularly when no connection to the ClearQuest database could be established.



Beside the display of markers in the models, status information is also shown in the Matrix View of a Configuration Space. By double-clicking on the Config Space an editor opens and

Model Elements	Level	Indoor	IndoorDeluxe	Outdoor	OutdoorDeluxe	Thermometer
Weather Station						
Weather Station						
Debug	1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Trace	1.1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Output	2					
LCD	2.1					
PC Data transfer	2.2	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>
USB	2.2.1	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Serial	2.2.2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
Protocol	2.2.3	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>
SyndNGoProto	2.2.3.1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
UDPoverSLIPProto	2.2.3.2	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>
Sensors	3					
Pressure	3.1					<input type="checkbox"/>
Temperature	3.2					
Wind	3.3	<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>

Variant overview – View of the marker symbols in the Matrix View

shows one column for each variant. Each row corresponds to an element and the selection state for this element can be seen directly.

By selecting "*Visualize... -> ClearQuest Defects State*" in the context menu of the Matrix View, another view on the selections for the defect states of the elements can be shown. Thus it is evident which variants still have open defects. The symbols here are the same as in the models.

#### 4.4. Change the defect state in ClearQuest

Connector for ClearQuest synchronizes regularly with the ClearQuest database. Thus the current status information about defects is always shown and can be evaluated effectively and in a variant-specific way.

1. Execute a query in the ClearQuest Navigator to list the defects.
2. Change the state to another value by selecting "Change State" from the context menu of a defect. (E.g.: Set the defect to *closed/resolved* by selecting *Close* or *Resolve* and complete the *Resolution* e.g. fixed. Or reopen the defect with *Reopen/Reject* and reset the *Resolution* value vice versa)

For this example it may help to experiment a little with the conditions in order to see how the corresponding changes take effect in pure::variants.

## 5. Future development of the Connector

With Connector for ClearQuest it is currently possible to visualize the current states of defects and tests in a variant-specific way and to use this information for further purposes. By means of the extensive information pure::variants contains, it is planned in future versions to create the corresponding entries automatically in ClearQuest. Thus “stubs” of test cases could be generated in ClearQuest from feature and family models, in order to guarantee a minimum of test coverage.

A far more interesting and complete option would be the automatic generation of a majority of the elements in the ClearQuest testing hierarchy. All necessary information can be stored in pure::variants models to export all data, from configuration attributes, to configurations themselves, up to the configured test cases and the test suites into ClearQuest. Information about the execution of concrete test cases and test logs would still be managed in ClearQuest, however most of the structural information would be extracted from the variant management. In addition the test history as well as all information relevant for test execution is administered and made accessible in ClearQuest.

## 6. References

For an introduction to ClearQuest or pure::variants, the respective documentation and the beginners tutorials are recommended.

You will find more information and resources (documentation, tutorials, white papers and evaluation software) on the pure-systems and IBM Rational websites.

<http://www.pure-systems.com>

<http://www.ibm.com/developerworks/rational/>

<http://www.ibm.com/software/rational/>