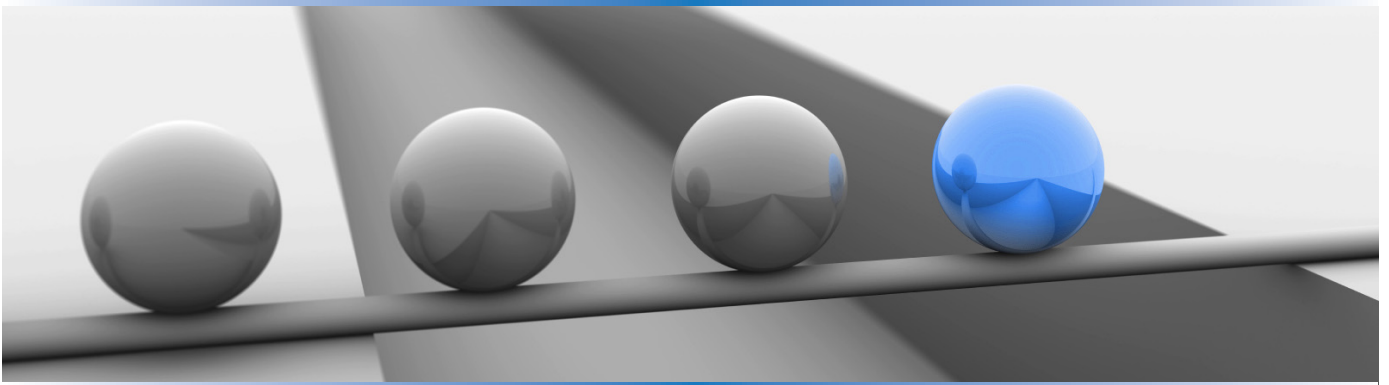


## Variability and Persistence



The newest additions to the QA vocabulary

**MKS**

## Executive Summary

The testing organization within any business is constantly under pressure to ensure that the delivered product or system meets certain quality goals (stated nor not) and to validate this under critical time pressures.

Testing is often one of the last activities in a development or engineering cycle purely due to the nature of evaluating constructed systems, and all too often an insufficient amount of time is set aside to adequately perform the desired level of testing. In order to rise to this challenge, testing teams must prioritize their work, evaluate risk and work with stakeholders to shift their testing efforts to the most pressing areas.

This paper describes how the increasing complexity of software can derail the testing organization and offers specific strategies that a QA team can use to overcome product or system complexity.

## Variability (the new reuse) - Variability (friend or foe)

Whether you call it Service Oriented Architectures, Component Based Development or Software Product Lines, variability in product or system development is gaining momentum in all classes of development groups. Engineering companies are trying to rapidly innovate through the introduction of product lines and product variants, IT companies are trying to rationalize a range of front office and back office systems that have been acquired through M&A activity or years of fast paced releases. The result in both scenarios is greater variability in each and every software release.

What does this mean for the Quality Assurance organization? It means more testing paths, more complex configurations and it usually means that QA becomes a bottleneck in getting releases out the door. Some QA teams will have a tendency to battle against variability, negotiating with engineering teams the number of supported platforms, configurations, concurrent product lines and product launches – however, when time to market is critical, the QA team will need to adapt.

For years software development teams have been adapting to variability, through architectural design patterns, software reuse strategies and many lessons have been learned along the way. If we reflect on some of those lessons we can gain a new appreciation for how we can deal with variability in test case design and development.

## Persistence; existing for a long or longer than usual time or continuously

Until now, tools and practices supporting QA have hampered the ability to design and create persistent, reusable test assets. The use of tools like Microsoft Word and Excel have done nothing to help \_ they promote the practice of rewriting test cases or at best, cloning a copy of those test cases.

When dealing with complex products and systems that need to be tested, we need to consider the persistence of our test assets. What do we mean by the persistence of test artifacts? It means rather than starting from scratch each time a new project or release is initiated, we stop and take stock of what we already have and how we can use those assets to our advantage.

There are several strategies for achieving persistence.

## Achieving Persistence through Copy

One of the more popular environments for authoring test cases and reporting on testing progress is Microsoft Excel. Excel offers some unique advantages with respect to sorting, filtering and manipulating large sets of data. There are two specific challenges with this technology for authoring test artifacts. First, it is difficult for a team to collaborate and share information with stakeholders and second, reuse is only really possible through a philosophy of copying or cloning.

Copying is the way in which most organizations today adapt to the need for persistent test cases. Test cases are written once for a specific project or release and then at a tactical level we try to copy/paste much of the valuable test design detail from past projects when moving forward. The problem with this approach is that we end up with many different versions of test cases all of them changing slightly and it does nothing to create a sense of value in our test assets.

After the third project or release we start to look back and say, where should I be copying my test assets from, the previous project?

## Achieving Persistence through Reuse

An improved approach to persistence is to develop a formal re-use strategy with a system that supports the reuse of test cases through a reference model. Rather than copying or cloning test cases, we refer to or reference a singular test case in a library.

This model is much stronger as it encourages the common use of an asset and the community that subscribes to that test asset has a motivation for contributing value to that library. We can evolve the singular test case over time, keeping it modernized and practically useful.

Achieving reuse in this way is not without its challenges. It requires an organization to be much more mature about how they capture test cases and control them. Configuration management practices are required so that we can baseline the library of test cases and identify versions of tests that are used in a given project. Change management controls are also required in order to ensure that any change made to a shared test case is done in a controlled, approved manner that is consistent with the goals of that shared artifact. Test designers may need specialized training in order to understand the development of abstract test cases and will inevitably need to work more closely with the architects and development teams constructing the product or system.

The final challenge for organizations that are testing in a reuse paradigm is the proliferation of test cases required for each requirement or aspect of variability in a system or product.

## The Role of Parameters in Persistence

In addition to the foundations for supporting reuse (configuration management), we also need to encourage organizational practices that will ensure reuse is a realistic possibility. For example, simply having the configuration management capabilities available to support reuse, doesn't mean test designers or developers will actually produce test cases which can be effectively reused. Part of this happens with proper team mentoring and organizational training, but another part requires a system which can aid in making test artifacts more generic and therefore more reusable. As with any reuse strategy, variability is the key.

How do we achieve variability in any engineering artifact or work product? Typically variability is achieved by creating abstract artifacts which can be parameterized. Test artifacts are no exception, they should support parameterization to enable more generic test artifacts to be written. When dealing with any aspect of variability, a testing organization should be working closely with the architects, business analysts and development teams upstream to identify the variable aspects of the system under test. Test cases should be written using more abstract design patterns, lending themselves to the same variability patterns of Object Oriented Development.

Consider an example of an aerospace company which produces digital cockpit displays and information systems. In order for this company to be successful it needs to build and market products which are capable of operating in a variety of airplane makes, models and configurations. An engine monitoring information system may need to interface with engines produced by General Electric, Rolls-Royce or Bombardier on a variety of different airplanes from Boeing or Airbus targeted at many different customer airlines. In order for this company to be profitable, the products and systems need to be developed and architected in an intelligent way, where commonality is exploited.

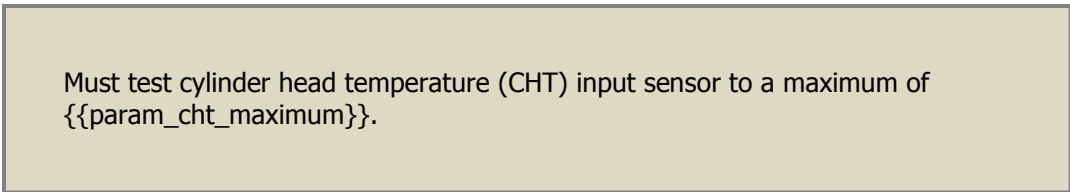
*"Product lines are, of course, nothing new in manufacturing. Boeing builds one, and so do Ford, Dell, and even McDonald's. Each of these companies exploits commonality in different ways. Boeing, for example, developed the 757 and 767 transports in tandem, and the parts lists for these very two different aircraft overlap by about 60%." – Software Product Lines Practices and Patterns, Paul Clements and Linda Northrop*

So considering the need for engineering teams to develop products with a reuse strategy, why should the QA and testing teams behave any differently? Clearly the answer is that they shouldn't.

So let's examine this scenario more closely and identify the roles that parameters play. Parameterization can be defined as a way to take certain aspects of a test case design and make it more generic.

If we consider the aerospace company, they may build an engine monitoring information system that needs to display the operating temperature of each cylinder head in the engine. This would be referred to as CHT (Cylinder Head Temperature). So this aerospace company will construct a base product with temperature sensor inputs capable of reading temperatures ranging from 0 Fahrenheit to a maximum of 1600 Fahrenheit. However, they may need to calibrate the software for different temperature ranges for different engines.

Figure 1 - An example of an abstract test case definition



Must test cylinder head temperature (CHT) input sensor to a maximum of {{param\_cht\_maximum}}.

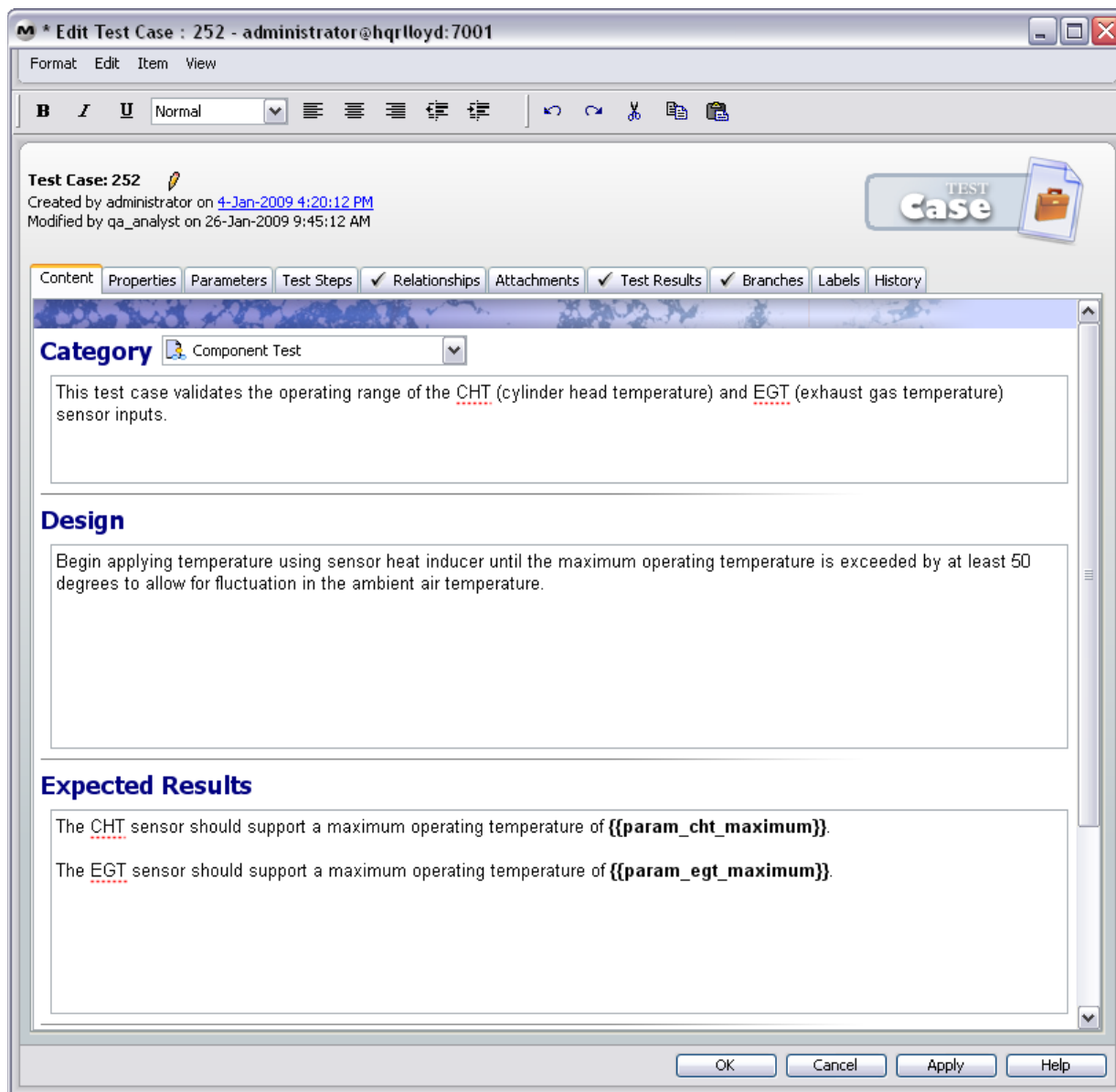
Then each time a release is produced for a specific engine specification, the test case can be varied and a parameter value specified. E.g. param\_cht\_maximum=1600F

## Creating a Test Case Library in MKS Integrity

### Designing a Parameter Reference

With MKS Integrity a parameter reference is the first step in creating a parameterized test artifact. When writing a test case or a test step we need to understand the items in the test artifact which may need variability. When constructing the test and stating the design or expected results, we can reference a parameter which would later be substituted with an appropriate value. We use special syntax to define parameters.

Figure 2 - Parameters can be referenced with special syntax in the design of the test case or test step

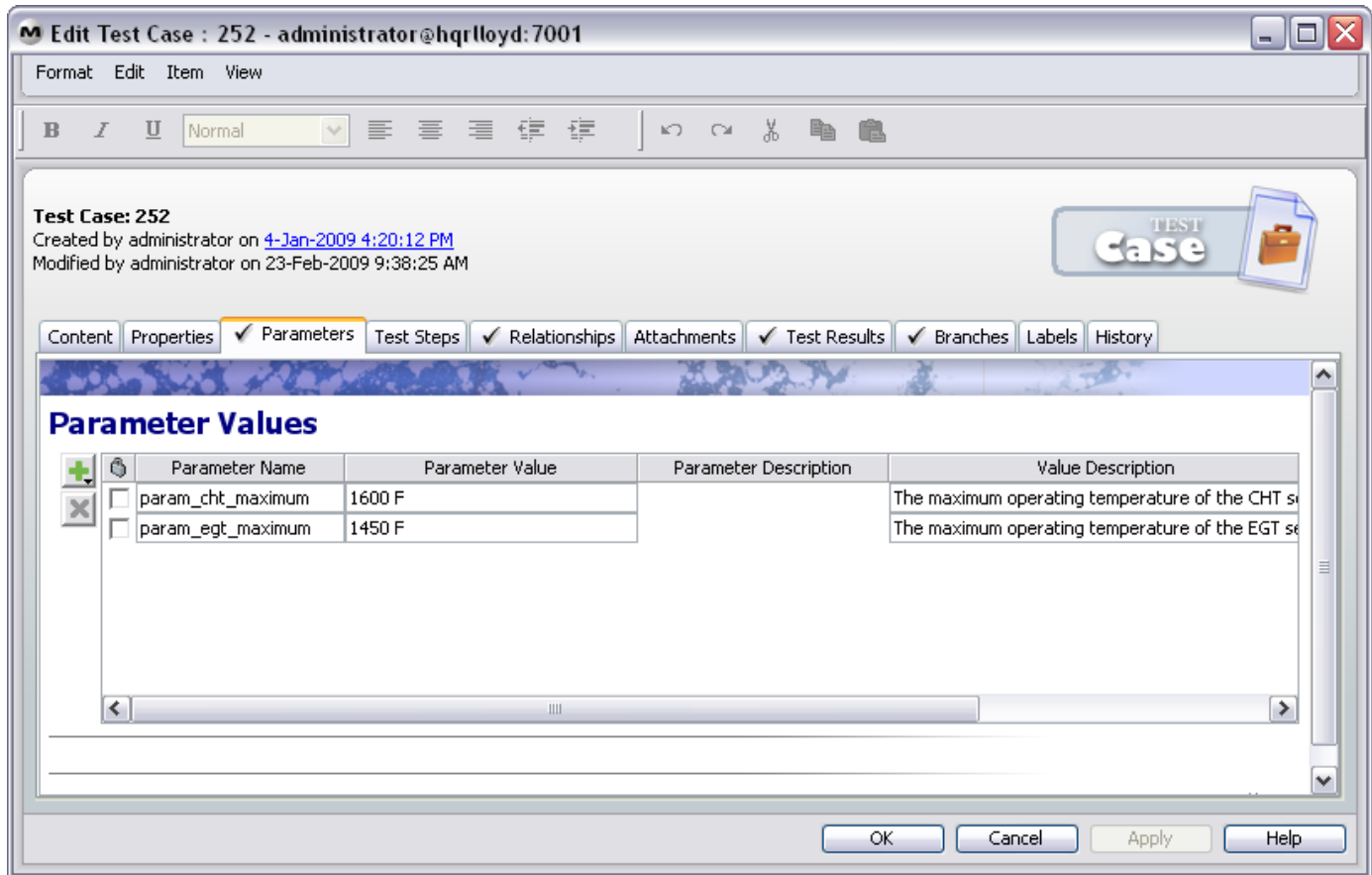


## Defining a Parameter Value

Parameter values can come from a variety of sources. The parameter value for a test step may be stated on the test case that incorporates that test step. In this fashion, every test case using a test step can vary the behavior of the step to suit their needs. In a case where a parameter is referenced in a test case, the value itself may come from the Project as a standard project parameter or perhaps the parameter value is specified at runtime by the test engineering performing the testing.

A key capability when setting parameter values at the project level is parameter enforcement through locking . By locking a project parameter, we enforce that all artifacts in the project referencing that parameter, adhere to that value and may not override it. In this way, a locking model or enforcement model for parameters becomes important. This ensures that project standards are adhered to, especially where parameters represent mission critical aspects of a system.

Figure 3 - Parameter values can be specified on the Test Case definition or at a Project Level



## Parameter Substitution

In addition to supporting the use of parameters from a documentation and design perspective, it is also important to appropriately substitute parameters when a test is being carried out. When executing a test case it is necessary to be aware of the actual parameter value that is relevant to the context. Contextual awareness can be derived from the Test Case definition, from the Project, or may be left to the test engineer to define at runtime. Parameter substitution ensures that the test engineer carrying out the test will see the appropriate substituted value for their parameter when the test is run.

## Summary

In order for testing teams to keep up with the rapid pace of application development they should look for support in building more reusable and persistent test assets, creating more value in the testing lifecycle.

MKS Integrity can improve the effectiveness of QA by providing capabilities to manage test cases persistently, reusing those valuable testing assets release over release and from product to product. Through strong support for the disciplines of change management, configuration management and parameterization, the MKS Integrity platform will help you transition the perception of QA from a bottleneck, to a QA team that is innovative and essential in ensuring software quality.

Figure 4 - Parameters may be substituted with their values when viewed or reported.

**Test Case: 252**  
Created by administrator on [4-Jan-2009 4:20:12 PM](#)  
Modified by administrator on 23-Feb-2009 9:38:25 AM

**TEST Case**

Content Properties  Parameters Test Steps  Relationships Attachments  Test Results  Branches Labels History

**Category** Component Test

This test case validates the operating range of the CHT (cylinder head temperature) and EGT (exhaust gas temperature) sensor inputs.

**Design**

Begin applying temperature using sensor heat inducer until the maximum operating temperature is exceeded by at least 50 degrees to allow for fluctuation in the ambient air temperature.

**Expected Results**

The CHT sensor should support a maximum operating temperature of **1600 F**.

The EGT sensor should support a maximum operating temperature of **1450 F**.

Close Help Print

## **MKS Headquarters**

### **North America**

1 800 613 7535

### **UK & Northern Europe**

44(0) 1483 733900

### **Central & Southern Europe**

49(0) 711 3517750

### **Denmark**

+45 4420 9831

### **Singapore**

65 6732 8768

### **Japan**

03 5789 5544

**[sales@mks.com](mailto:sales@mks.com)**

For more information

visit **[mks.com](http://mks.com)**

# **MKS**