

Category Partition Method (CPM)

An Approach Paper on Applying CPM to Mobile Platform Testing

A SIPTECH WHITE PAPER

Version 1.04

Inside this document ...

- 01. Introduction
- 02. Background
- 03. Category Partition Method
- 04. CPM Pilot Implementation
- 05. Analysis
- 06. Limitations
- 07. Inferences
- 08. Next Steps

© 2005 All rights reserved. SIP Technologies and Exports Ltd. The information contained in this document is CONFIDENTIAL and PROPRIETARY in nature, and subject to the rights and ownership of SIPTECH. Any and all unauthorized copying or use of the contents hereof is prohibited.

Acronyms, Abbreviations and Symbols

Acronym	Definition
CPM	Category partition method
TCD	Test case definition
TCDRT	Test case definition review time
TCDT	Test case definition time

References

T. J. Ostrand, and M. J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests", Communications of the ACM, vol. 31, no. 6, pp. 676--686, June 1988.

A. J. Offutt & A. Irvine, "Testing object-oriented software using the category-partition method." Proc. Seventeenth Int. Conf. on Technology of OO Languages and Systems (TOOLS USA'95), Santa Barbara, California, USA, pp. 293-303, August 1995.

Using Formal Methods To Derive Test Frames in Category-Partition Testing. Paul Ammann and Jeff Offutt. Ninth Annual Conference on Computer Assurance (COMPASS 94), pages 69-80, Gaithersburg, Maryland, June 1994.

Irvine & A. J. Offutt, "The effectiveness of category-partition testing of object-oriented software." Tech Report ISSE-TR-95-102, Dept of Information and Software Systems Engineering, George Mason University, 1995.

T.Y. Chen, P.-L. Poon, and T.H. Tse. A choice relation framework for supporting category partition test case generation. IEEE Transactions on Software Engineering, 29(7):577-593, 2003.

H.V. Kantamneni, S.R.Pillai and Y.K. Malaiya "Structurally Guided Testing," in Supp. Proc. Int. Symp. Software Reliability Eng., Nov 2002, pp. 137-138.

1. Introduction

Software testing is an extensive and difficult process for any realistic software system. It is even more so in the case of object oriented software and recent trends in component based development by making use of many third party components. By coming-up with a systematic test generation process, it is possible to have consistency and predictability across projects. This will also help automating the test generation process and thus the overall cost can be significantly reduced.

1.1 Component Technology Evolution

The evolution of software technology enables us to handle more complexity than ever before but also brings in associated testing challenges.

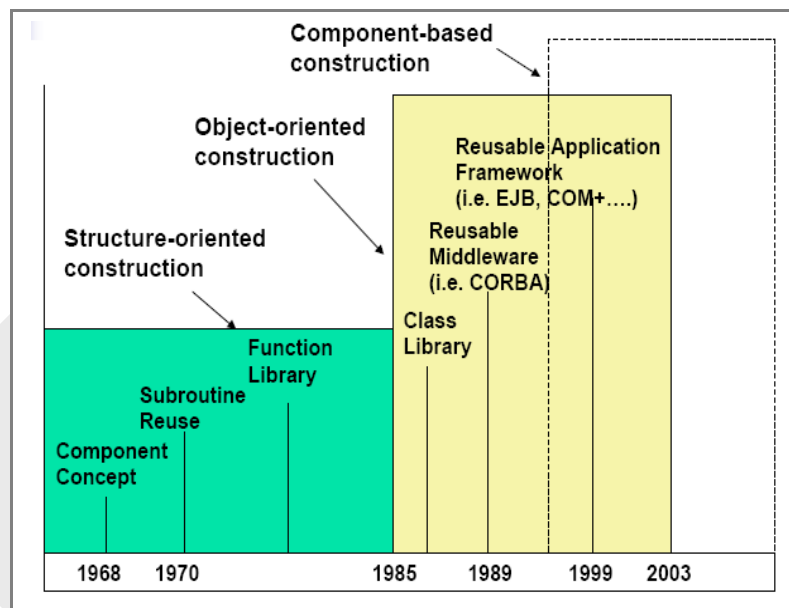
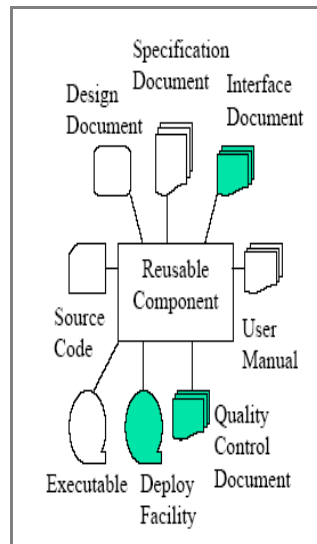


Figure 1

1.2 Software Component

A software component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties. - Clement Szyperski.



A software component differs from a software module in the following aspects:

- ❖ A component is developed for reuse by third parties
- ❖ A high quality component must be deployable, executable, independent, and well-interfaced.
- ❖ A reusable component usually can be reused in different reusable contexts and environments for different projects.

1.3 Component Testing Challenges

The components used for construction of software comes from many sources, say, third party developers, partners, other development groups within the same organization or from the earlier version of the software to name few sources. This poses the following testing challenges

- ❖ Difficult to perform component analysis and testing due to the lack of the access to component source code and internal artefacts. (Poor testability)
- ❖ Testing reused components in a new reuse context and environment.
- ❖ Expensive cost of constructing component test environments

1.4 Testing component based software

It is often difficult to achieve adequate testing of component-based software for the following reasons:

- ❖ Existing testing techniques and methods do not consider adequate testing for software reuse
- ❖ Hard to measure if a component is adequately tested.

2. Background

As the development approach has moved from conventional to object-oriented programming and then onto component-based development, developers face difficult decisions in modifying their development process to best use the new technology and their testing process to ensure that the software is highly reliable in this new environment.

This poses challenges in evolving and implementing effective ways to test the object-oriented software to achieve test effectiveness coupled with coverage with an optimal test suite.

Much of the research on testing object oriented software try to answer the following questions [2]:

- ❖ How the properties of object-oriented software can be used to reduce the effort required to test software using object-oriented technologies?
- ❖ How to test object oriented software effectively?

There is a lot of literature on various testing techniques. A brief discussion on some of the more relevant approaches is presented by H.V. Kantamneni et al. [6]

Software testing techniques can be classified into two broad categories:

- ❖ Black box testing and
- ❖ White box testing

Black box testing techniques rely on abstract descriptions of the software such as specifications or requirements. White box testing techniques rely on the structure of the software and make use of the knowledge of the code.

2.1 Testing Techniques

A set of techniques that fall into either categories of testing is listed in the next section. The list is not exhaustive and presented here as an indication of choices available to the testers.

2.1.1 Black-box testing techniques

Black-box testing techniques can be classified into three broad groups:

Usage-based black-box testing techniques – focusing on user-oriented accesses

- ❖ User-operation scenario testing
- ❖ Random testing
- ❖ Statistical testing

Error-based black-box testing techniques – focusing on error-prone points

- ❖ Equivalence partitioning testing
- ❖ Category-partition testing
- ❖ Boundary-value analysis
- ❖ Decision table-based testing

Fault-based black-box testing techniques – focusing on targeted fault points

- ❖ Fault injection method

2.1.2 White box testing techniques

- ❖ Path analysis
- ❖ Data flow testing
- ❖ Domain testing
- ❖ Mutation testing

2.2 Why Category Partition Method

Category partition method has been recognized as a valuable testing technique for specification based testing of both functional and API testing.

The existing organisational testing process depends on the use of equivalence partitioning and boundary analysis techniques. To bring in a higher level of consistency and predictability in the test identification process towards meeting the test effectiveness and coverage goals, Category partition method has been identified as a candidate technique for piloting in an API test suite development project.

3. Category Partition Method

CPM is a specification-based testing technique developed by Ostrand and Balcer [1]. It helps software testers create test cases by refining the functional specification of a program into test specifications. It identifies the elements that influence the functions of the program and generates test cases by methodically varying these elements over all values of interest. The method consists of the following steps:

1. Decompose the functional specification into functional units that can be tested independently.
2. Identify the parameters (the explicit inputs to a functional unit) and environment conditions (the state of the system at the time of execution) that affect the execution behavior of the function.
3. Find categories (major properties or characteristics) of information that characterize each parameter and environment condition.
4. Partition each category into choices, which include all the different kinds of values that are possible for that category.
5. Determine the constraints among the choices of different categories. For example, one choice may require that another is absent, or has a particular value.
6. Write the test specification (which is a list of categories, choices, and constraints in a predefined format) using the test specification language TSL.
7. Use a generator to produce test frames from the test specification. Each generated test frame is a set of choices such that each category contributes no more than one choice.
8. For each generated test frame, create a test case by selecting a single element from each choice in that test frame.

An important aspect of category-partition testing is the construction of test specifications as an intermediate between functional specifications and actual tests.

The goal is to apply category partition technique to come-up with a systematic process to produce a test specification that satisfies the coverage and test effectiveness criterion. As the category partition method is known to throw up more combinations the process should also address the issues of constraining the combinations within practical limit, problem of resolving infeasible combinations of choices for categories.

4. CPM Pilot Implementation

The first pilot focuses on the usefulness of CPM for test generation for API testing. In an API testing scenario, CPM will have a lot of commonality with equivalence partitioning and boundary analysis.

4.1 Pilot Background

For this pilot, about six API methods from a cross section of classes from a platform development project have been chosen. The APIs are implemented in Java.

An iterative development model is being used as the lifecycle model with each iteration spanning four weeks on an average. Each iteration consists of new feature development, bug fixes by the development team and the parallel development of tests for new features, maintenance of existing tests due to changes to already

implemented features, multiple regressions runs on intermediate builds until a stable build qualifies for release into integration testing.

This process leaves the testing team with limited time for coming-up with new tests to meet the coverage goals and test effectiveness. Given the challenges both in terms of coming-up with new test cases and maintaining already developed tests with minimal effort, it is imperative to have a more systematic way.

The existing process based on equivalence partitioning and boundary value analysis techniques is less formal and relied on experience for meeting the coverage and effectiveness goals. With a view to improve the formalism of the process for better predictability and consistency on the one hand and general applicability of the process to specification based functional testing on the other hand, CPM was identified as a technique of choice for evaluation. Another goal was to analyze the feasibility of automation of all or part of the steps and plan for toolsets for the same in later phases.

4.2 Approach

Category Partition Method is a specification based technique and is language or implementation independent. It is based on the assumption that all errors are related to value combinations of various input parameters and instance variables that have global scope.

The approach adopted for the pilot is described below with examples:

4.2.1 Identify the parameters of the function

Given the fact that this pilot starts at the level of testing the functionality at method level, the first step of the CPM method is unnecessary i.e. decomposing the functional specification into functional unit is not required, as the methods constitute the functional units.

Specifically, method interface arguments (input and output parameters) and abstract class state (system, global or environment variables) that determine the function's response are identified.

EXAMPLE

The API method `FileFolder.getEntries(int startPosition, int length)` returns an array of entries. The entries in the folder from the start position for the length are returned.

The two explicit parameters that influence the outcome of the function are start position and length. The environment or global variable that would affect the outcome would be the number of entries in the folder. Thus there are three identified parameters.

Class : FileFolder

Method : getEntries(int startPosition, int length)
 Return : Entry[]
 Exception : FolderException - Invalid start position or Length
 Parameters : int startPosition - must be >=1
 int length - must be > 0
 No of entries in the folder - must be >= 0

4.2.2 Identify categories for the parameters

A category is defined as a subset of parameter values that determines the behaviour of the function and is not part of any other category for that parameter. Categories are conceptually similar to equivalence classes.

EXAMPLE

The various categories identified are: -

Parameter	Category
startPosition	$n < 1$
	$1 \leq n < \text{Size of folder}$
	$n \geq \text{Size of folder}$
Length	$n \leq 0$
	$0 < n \leq \text{Size of folder}$
	$n > \text{Size of folder}$
No of entries in folder	$n < 0$
	$n = 0$
	$n > 0$

4.2.3 Partition each category into choices.

A choice is a specific test value for a category. Any reasonable criteria or historical data can be used to make a choice. Choices are normally made on the boundary conditions. Choices should be both valid and invalid values so as to exercise error and exception handling.

EXAMPLE

The various choices identified are:-

Parameter	Category	Choices
startPosition	$n < 1$	0
	$1 \leq n < \text{Size of folder}$	1, $n < \text{Size}$, $n = \text{Size} - 1$
	$n \geq \text{Size of folder}$	$n = \text{Size}$, $n > \text{Size}$

Length	$n \leq 0$	$n < 0, n = 0$
	$0 < n \leq \text{Size of folder}$	$1, n < \text{Size}, n = \text{Size}$
	$n > \text{Size of folder}$	$n > \text{Size}$
No of entries in folder	$n < 0$	-1
	$n = 0$	0
	$n > 0$	Size

4.2.4 Identify constraints between choices

Some choices are mutually exclusive or inclusive. Sometimes one choice made to one parameter precludes all the other choices of a parameter. For e.g. while processing claims, the number of claims or claim amount does not matter when the policy is cancelled.

This step is the most critical step in CPM and is also the most subjective part of CPM. To eliminate subjectivity a repository of parameters, categories and choices made should be built which can be referred upon while generating new test suite.

EXAMPLE

The various constraints identified between the choices are:-

Constraint 1: When the number of entries in the folder is -1, it precludes a response for $n < \text{Size}$, $n = \text{Size}$ and $n > \text{Size}$ choices of startPosition

Constraint 2: When the number of entries in the folder is 0, it precludes a response for $n < \text{Size}$, $n = \text{Size}$ and $n > \text{Size}$ choices of startPosition

Constraint 3: When the number of entries in the folder is -1, it precludes a response for $n < \text{Size}$, $n = \text{Size}$ and $n > \text{Size}$ choices of length

Constraint 4: When the number of entries in the folder is 0, it precludes a response for $n < \text{Size}$, $n = \text{Size}$ and $n > \text{Size}$ choices of length

4.2.5 Enumerate all choice combinations and expected results

The test case definition or matrix is generated by making a cross product of all choices of parameters. Mutually inclusive or exclusive constraints identified will lead to some test cases being dropped from the cross product set.

After developing the matrix or test case definition fill out the expected results for each of the choice combination.

EXAMPLE

Test Case	Function Parameters/Choices			Result	
	No of entries in Folder	startPosition	Length	Return	Exception
1	n=Size	n=0	n<0		FolderException: Invalid start position
2	n=Size	n=0	n=0		FolderException: Invalid start position
3	n=Size	n=0	n=1		FolderException: Invalid start position
4	n=Size	n=0	n<Size		FolderException: Invalid start position
5	n=Size	n=0	n=Size		FolderException: Invalid start position
6	n=Size	n=0	n>Size		FolderException: Invalid start position
7	n=Size	n=1	n<0		FolderException: Invalid length
8	n=Size	n=1	n=0		FolderException: Invalid length
9	n=Size	n=1	n=1	First entry	
10	n=Size	n=1	n<Size	Entries from 1 to specified length	
11	n=Size	n=1	n=Size	All the entries in the folder	
12	n=Size	n=1	n>Size	All the entries in the folder	
13	n=Size	n<Size	n<0		FolderException: Invalid length
14	n=Size	n<Size	n=0		FolderException: Invalid length
15	n=Size	n<Size	n=1	The specified entry in the start position	
16	n=Size	n<Size	n<Size	Entries from specified start position for specified length	

17	n=Size	n<Size	n=Size	All the entries from the specified start position	
18	n=Size	n<Size	n>Size	All the entries from the specified start position	
19	n=Size	n=Size-1	n<0		FolderException: Invalid length
20	n=Size	n=Size-1	n=0		FolderException: Invalid length
21	n=Size	n=Size-1	n=1	The last entry	
22	n=Size	n=Size-1	n<Size	The last entry	
23	n=Size	n=Size-1	n=Size	The last entry	
24	n=Size	n=Size-1	n>Size	The last entry	
25	n=Size	n>Size	n<0		FolderException: Invalid length
26	n=Size	n>Size	n=0		FolderException: Invalid length
27	n=Size	n>Size	n=1	Null	
28	n=Size	n>Size	n<Size	Null	
29	n=Size	n>Size	n=Size	Null	
30	n=Size	n>Size	n>Size	Null	
31	n=0	n=0	n<0		FolderException: Invalid start position
32	n=0	n=0	n=0		FolderException: Invalid start position
33	n=0	n=0	n=1		FolderException: Invalid start position
34	n=0	n=1	n<0		FolderException: Invalid length
35	n=0	n=1	n=0		FolderException: Invalid length
36	n=0	n=1	n=1	Null	
37	n<0	n=0	n<0		FolderException: Folder does not exist
38	n<0	n=0	n=0		FolderException: Folder does not exist
39	n<0	n=0	n=1		FolderException: Folder does not exist

40	n<0	n=1	n<0		FolderException: Folder does not exist
41	n<0	n=1	n=0		FolderException: Folder does not exist
42	n<0	n=1	n=1		FolderException: Folder does not exist

5. Analysis

The project described in the problem statement is made up of multiple components. Two of the modules under the project were taken up for piloting CPM. The core team from Testing Strategy and Practice took up the study of the technique and published the procedure for CPM and template to develop the CPM matrix.

The project lead then with the help of Testing Strategy and Practice team came up with metrics to be measured to capture the impact of CPM if any on project life cycle. The basic metrics that were defined to be captured and compared with heuristic approaches are:

- ❖ Test Case Definition Time (TCDT) = Actual efforts spent on Test Case Definition/No of test cases developed in an iterative cycle.
- ❖ Test Case Review Time (TCDRT) = Actual efforts spent Review of Test case definition/No of test cases reviewed in an iterative cycle.
- ❖ Code Coverage = % of lines executed is obtained by using code coverage tools.
- ❖ Cost of Testing = Total time spent on testing/Number of bugs reported.

The performance metrics data for iterative cycles prior to implementing CPM was consolidated for the identified components to establish the benchmark against which the data collected from piloting CPM would be compared and analyzed against.

5.1 Test Case Definition Time

TCDT is defined as the ratio between actual efforts spent on creating test case definition (TCD) to the total number of test cases defined in an iterative cycle.

The data collected for the two components prior to piloting CPM has been tabulated below.

	Component 1			Component 2		
	# of TCD	Efforts (hrs)	TCDT	# of TCD	Efforts (hrs)	TCDT
Iteration 1	72	35	0.49	74	24	0.32
Iteration 2	59	29	0.49	48	18	0.38
Iteration 3	0	0	NA	0	0	NA
Iteration 4	67	31	0.46	79	32	0.41
Iteration 5	29	16	0.55	85	27	0.32

The focus of Iteration Cycle 3 was on performance improvements and no new features were added. Hence, the entire cycle was spent on automation and regression and performance improvement testing and no new tests were added. Therefore the data from that iteration has been excluded from the analysis to avoid skewing of results.

In Iteration Cycle 6 CPM was piloted. Between the two components there were three new features that were required to be developed in the cycle. Test case definitions were created using Category Partition Methodology as per the steps described in the prior section. The data on the tests and effort spent were captured and tracked using the organisational Project Management System including timesheet. The summarized data gathered from piloting CPM has been presented below.

	Component 1			Component 2		
	# of TCD	Efforts (hrs)	TCDT	# of TCD	Efforts (hrs)	TCDT
Iteration 6	48	16	0.33	108	25	0.23

A marked improvement was seen in the number of new tests identified for each of the features for the given amount of time. This has translated into a reduction in the average time taken to define a test case. It is expected that there is scope for more improvement as this was a pilot effort where the learning time is significant. But over a period of time its effect will be negligible leading to further improvement in the average time taken to define a test case.

5.2 Test Case Definition Review Time

Test case definition review time [TCDRT] is defined as the ratio between actual efforts spent on reviewing test case definition to the total number of test case definition reviewed in an iterative cycle.

The data collected for the two components prior to piloting CPM has been tabulated below.

	Component 1			Component 2		
	# of TCD Reviewed	Efforts for review (hrs)	TCDRT	# of TCD Reviewed	Efforts for review (hrs)	TCDRT
Iteration 1	72	16	0.22	74	10	0.14
Iteration 2	59	12	0.20	48	7	0.15
Iteration 3	0	0	NA	0	0	NA
Iteration 4	67	14	0.21	79	14	0.18
Iteration 5	29	7	0.24	85	11	0.13

Iteration Cycle 3 does not have any data as that particular iteration was meant for performance improvements and no new features were added. The entire cycle was spent on automation and regression and performance improvement testing.

In Iteration Cycle 6 CPM was piloted. Between the two components there were three new features that were required to be developed in the cycle. Test case definitions were created using CPM as per the steps described in the prior section. The time taken for review of the test cases defined was captured using the Timesheet tools. The data collected from piloting CPM has been tabulated below.

	Component 1			Component 2		
	# of TCD Reviewed	Efforts (hrs)	TCDRT	# of TCD	Efforts (hrs)	TCDRT
Iteration 6	48	5	0.096	108	8	0.08

This proved to be area where the benefit realized was immediate as the review was more objective. Most of the review findings were in the identification of choices and constraints. There is scope for more improvement as CPM is adopted for future iteration cycles and a repository of all the categories, choices and constraints relating to parameters are built to aid the same.

5.3 Code Coverage

Code coverage is measured by using CLOVER (code coverage tool). Code coverage is one of the most important means of validating the completeness of the test suite. Code coverage is basically the % of lines in the code being tested that was actually executed by the test suite during testing.

To effectively bring out the benefits of CPM apart from just piloting it in the iterative cycle no 6 we also took up some of the features covered in previous iterative cycles and developed a new test suite based on CPM for those features. After developing the same the code coverage for that release was measured and documented.

	Code Coverage % for Component 1		Code Coverage % for Component 2	
	Heuristic	CPM	Heuristic	CPM
Iteration 1	72	NA	81	NA
Iteration 2	74	NA	83	NA
Iteration 3	NA	NA	NA	NA
Iteration 4	69	NA	80	NA
Iteration 5	72	74	82	84
Iteration 6	NA	75	NA	84

The code coverage shows marginal improvement at the global level as the code coverage obtained with the tool was for the entire API and CPM was implemented only for selected features of the API. The potential of CPM will be more conclusive if CPM is used for all the new features in future iterative cycles and if all the existing features are revisited to apply CPM to redesign the test suite.

6. Limitations

As with other methodology or techniques, CPM also has its own advantages and limitations. This section presents the limitations of CPM and the major issues that were faced while piloting the same in an API testing scenario.

The major limitation of the CPM is that it does not unearth bugs due to stress or system errors. But when the scope of testing is just limited to functionality testing as in this case, CPM proves to be a very useful technique.

The major issue in implementing CPM is the size of the initial test suite generated. It is huge to be fully implemented for complex functionalities. If we take an example method with 5 parameters and two global variables, then even a minimum of two possible values per parameter would yield a test suite of cross-product of all choices which would be around 128 test cases. But in a realistic example, the range of possible values per parameter in a method is much more and thus the potential number of tests will run in to thousands.

Although this test count can be constrained by applying constraints based on parameter dependencies, the potential test suite size is still large. In such cases a complementary technique to limit the choices should be used. One important consideration for the choice of the complementary technique is the feasibility of automation, and less reliance on experience or subjectivity in imposing constraints. Techniques such as Orthogonal Arrays can be considered as options.

The second most prominent issue that we ran in to was the subjectivity involved in partitioning the category into choices. The choices differed from team member to team member based on their experience and other factors. This issue can be bought under control by rigorous reviews and maintaining a repository of choices and categories identified for parameters.

7. Inferences

CPM has been found to provide relatively less experienced testers, a systematic way to come-up with comprehensive test suite and help them find more defects in that process. Given the number tests it throws up, there is scope for constraining the test suite size without reducing the coverage or test effectiveness in uncovering defects.

Given the goal of having a systematic test generation process that can be applied consistently across different testing projects, there is scope for employing CPM technique. Its promise for automation with a suitable set of tools also adds to its potential usefulness in bringing down the overall testing effort leading to cost savings.

8. Next Steps

CPM paves way for automation of generating the tests for APIs at method level. A tool can be devised where the details of commonly used parameters and possible data ranges are already stored. This tool can take parameters of an API method as input and present an UI with matching categories from the database to allow the selection of applicable categories and generate a range of choices.

This selection of choices can then be reviewed in the context of known constraints. On completion of the above step an entire matrix with all possible combination of choices of various parameters can be exported to a spreadsheet. This can serve as the input for the identification of potential test cases that need to be developed.

9. About us

SIPTECH provides world class software testing and quality assurance services based on proven methodologies and processes. We focus on companies that produce software products and technology platforms. With SIPTECH you will reduce your costs, improve your quality, and benefit from our independence. We care about only one thing: helping you release and maintain a high quality product.

For more information about SIPTECH and its capabilities visit the World Wide Web at <http://www.siptech.com> and email us at info@siptech.co.in and sales@siptech.co.in.