Automated Unit Testing with ABAP

A Practical Approach

—

James E. McDonough

Apress®

Automated Unit Testing with ABAP

A Practical Approach

James E. McDonough

Apress®

Automated Unit Testing with ABAP: A Practical Approach

James E. McDonough
Pennington, NJ, USA

ISBN-13 (pbk): 978-1-4842-6950-3 ISBN-13 (electronic): 978-1-4842-6951-0
https://doi.org/10.1007/978-1-4842-6951-0

Copyright © 2021 by James E. McDonough

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Coordinating Editor: Rita Fernando

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484269503. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To Norman Edge, teacher and bassist extraordinaire, whose benevolent guidance toward improving the qualities of my musicianship also had a profound influence on the qualities of my character.

Table of Contents

About the Author xi

About the Technical Reviewer xiii

Acknowledgments xv

Chapter 1: Introduction 1

For Whom This Book Is Applicable 1

How This Book Should Be Used 2

Why This Book Was Written 3

Credentials of the Author 6

Summary 7

Chapter 2: Preparing to Take the First Step 9

Road Map to Automated Unit Testing 9

Where We Are Now 10

Where We Are Going 11

Why We Are Going There 11

How We Are Going to Get There 13

Legacy Code 16

Calisthenics 17

Summary 18

Chapter 3: Software Quality 21

The Quality of Software 21

Assessing Software Quality 24

Summary 27

Chapter 4: The Origins of Automated Unit Testing 29

In the Beginning … 29

The Emergence of xUnit 30

Features of xUnit 30

Phases of xUnit Tests 35

Writing xUnit Tests 36

Advantages of xUnit Tests 39

Summary 40

Quiz #1: xUnit Concepts 40

Multiple Choice: Select the Best Answer 41

True or False 42

Chapter 5: Automated Unit Testing with ABAP 43

ABAP Unit 43

The ABAP Unit Testing Framework 44

Requirements for Writing ABAP Unit Tests 46

Types of Components Applicable to Unit Testing 48

Testable ABAP Modularization Units 49

Automatic Generation of ABAP Unit Test Classes 51

ABAP Language Statements Related to Unit Testing 53

Writing an ABAP Unit Test 55

Using Fixture Methods 67

Invoking the Services of the ABAP Unit Testing Framework 71

ABAP Unit Test Runner 76

Unit Test Results Report 77

Initiating ABAP Unit Test Execution 79

Initiating Unit Tests from Within an ABAP Editor 79

Initiating Unit Tests from Outside an ABAP Editor 80

Evolution of the ABAP Unit Testing Framework 82

Challenges to Effectively Testing ABAP Code 83
Challenges Presented by Classic ABAP Event Blocks 84
Challenges Presented by Global Variables 84
Challenges Presented by the MESSAGE Statement 85
Challenges Presented by ALV Reports 87
Challenges Presented by Classic List Processing Statements 87
Challenges Presented by Open SQL Statements 89
Controlling the ABAP Unit Testing Framework 89
Client Category 89
Client Category Override 90
Unit Testing Configuration 91
Summary 95
Quiz #2: ABAP Unit Testing Concepts 96
Multiple Choice: Select the Best Answer 97
True or False 98

Chapter 6: Rudiments 99

Introducing a Simple Unit Test 99
Expanding Unit Test Coverage 103
Implementing Unit Tests for Function Modules 107
Implementing Unit Tests for Global Classes 108
ABAP Statements and Features Affecting Automated Unit Testing 110
Exploring the Effects of the MESSAGE Statement 110
Exploring the Effects of ALV Reports 115
Exploring the Effects of Classic List Processing Statements 119
How Automated Unit Testing Enables Confident Refactoring 122
Diagnosing the Absence of Sufficient Test Data 125
Creating and Using Fabricated Test Data 127
Gaining Control Over References to Modifiable Global Variables Within Subroutines 131
Summary 134

Chapter 7: Design for Testability 135

Changing the Production Path to Enable Automated Testing 135

Categorizing Input and Output 138

Encapsulating Indirect Input and Output 140

Interaction Points 143

Encapsulating Indirect Input Processes to Accommodate Unit Testing 149

Encapsulating Indirect Output Processes to Accommodate Unit Testing 153

Summary 158

Chapter 8: Test Doubles 159

Depended-On Components 159

The Purpose of Test Doubles 161

Alternative 1 163

Alternative 2 165

Alternative 3 166

Using Test Doubles 170

Test Double Using Base Class 171

Test Double Using Interface 181

Categories of Test Doubles 190

Using Test Doubles for Indirect Input 192

Using Test Doubles for Indirect Output 199

Summary 210

Chapter 9: Service Locator 211

Purpose of a Service Locator 211

Using a Service Locator 212

Using a Service Factory 226

Organizing Local Components 229

Summary 249

Chapter 10: Leveraging the Service Locator 251

Issues Requiring Leverage 251

Using the Service Locator to Manage Global Classes 253

Using the Service Locator to Manage Function Modules 255

Using the Service Locator to Manage MESSAGE Statements 261

Handling MESSAGE Statements Triggering Unconditional Unit Test Failures 262

Handling Unit Test Failures Arising from MESSAGE Statement Control Flow 274

Using the Service Locator to Manage List Processing Statements 288

Summary 304

Chapter 11: Test-Driven Development 305

The TDD Cycle 305

The Three Laws of TDD 306

The Benefits of TDD 308

Following the TDD Cycle 309

Summary 327

Chapter 12: Configurable Test Doubles 329

Isolation Frameworks 329

mockA 330

ABAP Test Double Framework 330

Summary 351

Chapter 13: Obtaining Code Coverage Information Through ABAP Unit Testing 353

Code Coverage Metrics 353

Summary 355

Chapter 14: Cultivating Good Test Writing Skills 357

The Pillars of Good Unit Tests 357

Test Simplicity 358

Test Coverage 360

SAP Recommendations and Constraints When Writing Unit Tests 361

Tips for Writing Unit Tests 362
Issues Related to Testing Object-Oriented Code 365
Summary 366

Chapter 15: Welcome to Autropolis 367
One Small Step for Manual Toward Automated 367
The Right Tool for the Job 368
Resistance Is Futile 369
Becoming the Agent for Change 370
Go Forth and Automate 371
Summary 371

Appendix A: Requirements Documentation and ABAP Exercise Programs 373

Appendix B: Answers to Chapter Quizzes 375
Answers to Quiz #1 375
Multiple Choice: Select the Best Answer 375
True or False 376
Answers to Quiz #2 377
Multiple Choice: Select the Best Answer 377
True or False 378

Appendix C: Concepts Associated with Defining Local Test Classes 381

Index 387

About the Author

James E. McDonough received a degree in music education from Trenton State College. After teaching music for only 2 years in the New Jersey public school system, he spent the past 38 years as a computer programmer while also maintaining an active presence as a freelance jazz bassist between New York and Philadelphia. Having switched from mainframe programming to ABAP in 1997, he now works as a contract ABAP programmer designing and writing ABAP programs on a daily basis. An advocate of using the object-oriented programming features available with ABAP, he has been teaching private ABAP education courses over the past few years, where his background in education enables him to present and explain complicated concepts in a way that makes sense to beginners.

About the Technical Reviewer

Paul Hardy joined HeidelbergCement in the United Kingdom in 1990. For the first seven years, he worked as an accountant. In 1997, a global SAP rollout came along, and he jumped on board and has never looked back. He has worked on country-specific SAP implementations in the United Kingdom, Germany, Israel, and Australia.

After starting off as a business analyst configuring the good old IMG, Paul swiftly moved on to the wonderful world of ABAP programming. After the initial run of data conversion programs, ALV (ABAP List Viewer) reports, interactive Dynpro screens, and SAPscript forms, he yearned for something more and since then has been eagerly investigating each new technology as it comes out, which culminated in him writing the book *ABAP to the Future*.

Paul became an SAP Mentor in March 2017 and can regularly be found blogging on the SAP Community site and presenting at SAP conferences in Australia (Mastering SAP Technologies and the SAP Australian User Group annual conference), at SAP TechEd Las Vegas, and all over Europe at various SAP Inside Track events. If you happen to be at one of these conferences, Paul invites you to come and have a drink with him at the networking event in the evening and to ask him the most difficult questions you can think of, preferably about SAP.

Acknowledgments

I could not have done this project without the help of others.

I extend my gratitude to Chris Bostian, Larry Nansel, and Brian Brennan, who, after attending a presentation I gave on the subject of ABAP Unit testing to my colleagues in 2011, presented me with the opportunity to undertake a pilot project exploring how automated unit testing could be incorporated into the ABAP development process used at that site.

Thanks go to Dr. Juergen Heymann and Thomas Hammer, both of whom did a magnificent job of preparing and presenting the openSAP course **Writing Testable Code for ABAP** between March and May 2018, through which I realized that what I already knew on this topic was only the tip of the ABAP Unit testing iceberg.

I am very grateful to Paul Hardy for agreeing to undertake the task of reviewing the content of the book and for doing such a magnificent job at it, offering many suggestions for improvement.

Susan McDermott, Rita Fernando, and Laura Berendson, my editors at Apress Media, LLC, were of enormous help in guiding me through the publication process and resolving the technical glitches we encountered along the way.

Finally, it would have been much more difficult to complete this project without the love and understanding I received from my family for tolerating my absences during those long hours on weekends and holidays while I was secluded in deep thought about how to organize and present this content.

CHAPTER 1

Introduction

It is unlikely you still remember the first unit test you ever ran for an ABAP program you wrote. It is very likely you remember the most recent one. It is also very likely that the first and the last, and indeed all those tests in between, consisted of a manual effort executing the program over and over again using various combinations of values to insure the program produced the expected results. This seems to be the unit testing experience for the overwhelming majority of ABAP programmers, who remain pedestrians on the development superhighway when it comes to unit testing. For programmers coding in many other languages, it is commonplace for automated unit testing frameworks to be used as the vehicle whisking them along the software development expressway toward high-quality software.

Because so many ABAP-ers continue to use a unit testing process that is both horribly inefficient and woefully inadequate to the task, you might assume that there is no automated unit testing framework available to ABAP as there is for so many other languages. That would be a false assumption. Not only is there an automated unit testing framework for ABAP but unlike other languages it is seamlessly integrated into the development environment. It is known as the ABAP Unit Testing Framework, or simply ABAP Unit. It has been part of the ABAP tool set since 2004 but remains virtually unknown to many ABAP programmers.

For Whom This Book Is Applicable

This book is applicable to ABAP programmers having little or no familiarity with the concepts associated with automated unit testing for ABAP as well as to ABAP programmers who already are familiar with ABAP Unit testing but who want to explore further its testing capabilities. Though generally applicable to a wide range of programmers having various levels of experience writing ABAP code, from beginners to seasoned experts, and certainly to those who are familiar with object-oriented concepts,

© James E. McDonough 2021
J. E. McDonough, *Automated Unit Testing with ABAP*, https://doi.org/10.1007/978-1-4842-6951-0_1

it is particularly applicable to those ABAP programmers who have not yet become familiar with or comfortable using the object-oriented model for program design.

How This Book Should Be Used

This book is modeled on the "learn by doing" premise. Accordingly, **Appendix A** contains information about retrieving the requirements documentation for the accompanying comprehensive set of executable ABAP exercise programs, with each exercise program illustrating or reinforcing some new concept introduced in the book, from writing the most basic automated test to refactoring a program to enable comprehensive unit testing upon it. This provides for a multitude of options for using the book and doing the corresponding exercise programs, among them:

- Writing each new exercise program based solely on the information provided by the requirements documentation accompanying the collection of executable example ABAP exercise programs. This option might need an occasional supplement of performing comparisons of adjacent versions of the executable example ABAP exercise programs just to reinforce that the correct decisions have been made.
- Writing each new exercise program after looking at how the new concepts were implemented in the corresponding executable example ABAP exercise program. This option probably will require constantly performing comparisons of adjacent versions of the executable example ABAP exercise programs to identify the differences between them.
- Dispensing entirely with writing any code and simply relying on the corresponding executable example ABAP exercise programs to illustrate the implementation.

Consider the following before deciding among the options outlined here. Because there are more than 180 executable example ABAP exercise programs accompanying this book, the easiest of these options, by far, is the last one. It will allow you to proceed through the exercises at the quickest pace, reaching the last exercise program in the shortest period of time. Accordingly, this may be the most tempting option. However,

it is most probable that you will learn more about automated unit testing by choosing one of the preceding options. This is because those options will force you to think about what you are doing and to actually write the unit tests, enabling you to try various options with each new exercise so that you can explore the nuances of how automated unit testing actually works. The best way to sharpen your testing skills and to acquire the knowledge and wisdom necessary to implement comprehensive automated unit tests is to experience the satisfaction that comes with wrestling the code into submission by your own hand. Surely it will be more arduous and tedious, and certainly it will take longer to complete all the exercises, but in the end you will have become much more adept at making the decisions required to insure that your program is flexible, robust, and correct.

Refer to **Appendix B** for instructions for retrieving the accompanying collection of executable example ABAP exercise programs and their corresponding diagrams.

Why This Book Was Written

In January 2011, I became aware of ABAP Unit, the automated unit testing feature provided with SAP releases and available directly from the ABAP editor. I began to explore the possibilities of writing these *automated* unit tests for programs in the hope that I could present a convincing case to management for allowing them as an alternative to what until then had been rigid requirements for writing a formal unit test plan using a cumbersome spreadsheet template in which manually executed test results were to be recorded and then saved as a permanent artifact to accompany the software release documentation.

By then I had concluded that the manual spreadsheet process for testing, converted from a text document format well over three years before and taking far too long to execute a single unit test, was ineffective in assessing the quality of the software simply because it left too much to chance whether the actual test would sufficiently cover most parts of the software. Worse, the test plan, prepared by the developer who wrote the software, often was written in a way that assumed much application knowledge on the part of the person running the test, making it virtually useless to some other developer unfamiliar with that application, a discovery I made when I found and tried to run a unit test plan that had been written by someone else years earlier.

MY "AHA!" MOMENT WITH UNIT TESTING

I had first heard about the Agile software development philosophy in late 2008. Over the next year or so, I devoured many of the articles on this topic available on the Internet. This eventually led me to articles about Test-Driven Development (TDD). These TDD articles constantly stressed the importance of writing the unit test prior to writing the corresponding production code, but I found the explanations to be somewhat lacking because my concept of writing the unit test was modeled after the process used at my site, where the unit test was written using a spreadsheet or similar text document, to be executed manually once the production code became available. It was only after months of reading such articles that it finally dawned on me that the tests being discussed in these TDD articles were *automated*, tests that could be executed by the push of a button and run to completion in seconds. It was my "Aha!" moment to realize that the significant characteristic of these unit tests was that they were automated, a word that curiously had been missing from all those articles.

In April 2011, I prepared and presented to my development colleagues a demonstration on the benefits to be gained by using ABAP Unit testing. Also in attendance were some management personnel I had invited. At the conclusion of the demo, I was approached by a few of the managers who asked me whether I could devise a pilot project using software I already had been developing through the current project pipeline. I jumped at the opportunity and in July 2011 made another presentation to three representatives of the combined development and support staff illustrating the ease by which software already flowing through the development process could be thoroughly unit tested *automatically*. Sadly, this presentation was not well received by all who attended.

One point raised was that it took some time to write the code to run the automated unit test, time not currently budgeted with our current process. To this I responded candidly that, yes, it took longer to write the ABAP Unit test code than it would take to prepare an equivalent spreadsheet-based unit test, perhaps an order of magnitude of two or three times as long. However, I continued, although it might take longer to write ABAP Unit test code, there was significant time to be saved because it often took *hours* to run the spreadsheet-based test compared with only *seconds* to run the automated test, pointing out that repeated executions of the spreadsheet-based test would consume the same number of *hours* with each test execution compared with only the same number of *seconds* with each repeated execution of the automated test. Furthermore, in contrast

to the spreadsheet-based test plan usually glossing over specific application knowledge possessed by the person both writing and executing it, making it ill-suited for use by anyone other than the author, the automated unit test could be run by any developer, irrespective of their familiarity with the application.

Another point raised was that it would cause more work on the part of maintenance developers who now would need to learn to use this new capability and accommodate it in code where automated unit tests had been included by the original developer, going so far as to suggest that this should be considered grounds for not using automated unit tests. I did not challenge the point but merely agreed that, yes, it might require developers to learn and become more comfortable with this new feature. However, I was flabbergasted that anyone with the authority to manage a software development staff would raise what I considered to be such an indefensible position. Here was a representative of presumably capable developers insinuating that not only did those maintenance developers currently have no knowledge of ABAP Unit testing but neither should they be expected to make any effort to learn it.

Taking that reasoning to its logical conclusion, new software development should employ no technology, technique, or feature that might require a maintenance programmer to keep abreast of the technological improvements constantly being introduced in new releases. This would eliminate many features introduced in SAP releases *more recent than that familiar to the average maintenance developer* from ever being implemented in subsequent development efforts, from the simple use of ALV to the more advanced implications of using object-oriented design.

The three representatives in attendance would later consider the merits of my request for allowing ABAP Unit testing to be used as an alternative to the spreadsheet-based method already in place. About a week later, I received news of their final decision: not only would ABAP Unit testing *not be accepted* as an alternative to the current testing requirements but indeed the use of ABAP Unit testing *would be prohibited* for testing any code going to production. The reason for the draconian ruling, I was told, was due to concerns some of these managers had with perceived problems that ABAP Unit testing code introduces into the production environment.

Despite my attempt to improve the development process, I had succeeded in getting a useful automated testing tool blacklisted not only for myself but also for all the other members of my 30–odd person development staff. A few years later, another developer challenged the prohibition on the grounds that SAP itself recommended using ABAP Unit, and finally its prohibition was rescinded, but not before much new development

that might have gained some benefit from its use already had gone to production. I took advantage of the revised policy and gradually began to include ABAP Unit tests with some of my development efforts.

Over time I found that writing ABAP Unit tests helped me to write better production code. My initial approach to using ABAP Unit testing was to write all the production code first and then write the associated automated unit test code later, similar to the process already established at that site where writing the spreadsheet-based test would not be started until after all the production code had been written. What I found was that the attempt to retrofit an automated unit test to my newly completed production code exposed deficiencies in the way the production code was written, requiring that I refactor it to enable a clean test. In other cases, a retrofitted automated unit test I implemented would encounter failures that I suspect I never would have found had I used the spreadsheet-based approach in preparing the test. I soon came to appreciate both the improved thoroughness of testing and the beneficial implications on software design arising from the use of automated unit testing. It is through this book that I want to sing those praises loudly to the ABAP development community.

Credentials of the Author

My formal training in the data processing industry consists of one year at a community college learning mainframe languages (IBM assembler, COBOL, and PL/I) and, nearly 15 years later, a six-week seminar on ABAP programming. Compared with some of my colleagues over the years, I have very little formal training in computer programming. Indeed, the only formal training I had undertaken on the subject of ABAP Unit testing was to attend the openSAP course "Writing Testable Code for ABAP" offered online during March–May of 2018. Everything beyond what I learned in that course I learned on my own. So what makes me think I am qualified to teach anyone else about the associated concepts?

Prior to getting into the data processing industry in 1982, I earned a college degree in music education and taught instrumental music for two years in two different public school districts in the state of New Jersey. During my college years, I made an effort to learn and gain some modicum of proficiency with all of the band and orchestra instruments. My perception then was that I could be a better music educator by understanding more about the struggles students endure when they endeavor to learn

to play a musical instrument. How, I thought, could I presume to teach a seventh grader how to play the trombone if I were not able to play it myself?

This philosophy on education served me well those two years I taught in the public schools, and I have continued with this approach ever since. Accordingly, although my credentials in data processing may not be as impressive as those of some of my colleagues, my background as an educator enables me to perceive the problems students are likely to encounter when learning any new skill. So I have learned all I could about ABAP Unit testing, some through the openSAP course noted in the preceding text and some on my own, and over the past few years have been able to employ this feature with some of my ABAP development efforts. I believe that now, having gained a certain level of proficiency in this subject, I am ready to impart what I know to others who also wish to become familiar with this fascinating field of automated unit testing.

Summary

This chapter described how the book should be used as the reader is guided from a reliance upon manual testing of ABAP software to one based on automation. The audience is ABAP programmers. The approach to be used to convey the concepts is based on the "learn by doing" premise. In accordance with that premise, there are exercises the reader is urged to perform to reinforce those concepts, exercises based on a sizable collection of executable example ABAP exercise programs available for download. The reason for writing the book is based on the desire to share with others how they can reduce the time and effort involved in unit testing as well as to reveal the beneficial implications automated unit testing casts upon the design of software.

CHAPTER 2

Preparing to Take the First Step

Automated unit testing offers many new concepts for us to explore, so we will want to be certain we've taken the necessary precautions to insure a successful expedition into this new realm. Accordingly, let's take a moment to prepare ourselves for the adventure we are about to undertake, to pause and give consideration to both the journey itself and the expectations we have about what we will encounter along the way.

Road Map to Automated Unit Testing

A road map is a useful metaphor to illustrate the path we will take from our familiar surroundings of manual unit testing (MUT) to the unfamiliar new territory of automated unit testing. The road map shows us the way. We know we will need to travel the road between these two locations, eventually reaching our destination, but that each step along the way is dependent upon having taken the previous steps. That is, we move continuously in one general direction from our point of origin to our destination, covering each mile as we encounter it, not beginning to cover the tenth mile until after we already have passed through the ninth mile to get there. Accordingly, we become familiar with those parts of the road closest to our point of origin before those parts farther along. As with most such journeys, we find that the terrain associated with the first few steps is very similar to our starting location, but the terrain changes as we continue moving. This similarity of terrain between adjacent steps enables us to adapt gradually to the changes awaiting us along the road.

© James E. McDonough 2021
J. E. McDonough, *Automated Unit Testing with ABAP*, https://doi.org/10.1007/978-1-4842-6951-0_2

So, before we start on our journey from manual unit testing to automated unit testing, let us give some consideration toward preparing for a successful trip:

1. Where are we now?
2. Where are we going?
3. Why are we going there?
4. How are we going to get there?

Where We Are Now

If you are like many other programmers using SAP, you gained your experience writing ABAP programs before SAP introduced the feature known as ABAP Unit testing for facilitating automated unit testing of components written using the ABAP language, or if this feature had been introduced, your organization was not using a release where it was available to you. For most of us, the idea of unit testing never rose to the level of a topic worthy of education, training, and skill development during our careers, so generally we had been left to fend for ourselves when it came time to test a program. Over the years, each of us has collected useful techniques into our own personal bag of tricks to facilitate unit testing a program. Perhaps the one thing many of us share with each other is that unit testing has been and remains a dreaded and time-consuming *manual* process.

So here is where we find ourselves: capable ABAP developers, knowing very little about the new automated unit testing feature, ABAP Unit testing, and knowing even less about how to use it effectively to test ABAP components. It should come as no surprise that many ABAP programmers contemplating whether to learn and use this feature will choose to continue on with their manual testing techniques and avoid the automated unit testing feature so long as the standards in place at the site where they work do not require its use, but other ABAP programmers, who appreciate the significance of this new feature, who have become enlightened to the benefits of automating their unit testing efforts and want to leverage these new capabilities, will undertake to embrace this new feature and use it to their full advantage.

Where We Are Going

Automated unit testing facilities first emerged in the late 1980s when Kent Beck invented such a facility called SUnit for automatically unit testing programs written in the Smalltalk language and became further embraced in the late 1990s with the introduction of JUnit for automatically unit testing Java programs. Since then, a multitude of automated unit testing facilities have been created for various other programming languages.

In our quest to reach this district known as automated unit testing, we are headed for a place which was founded over a quarter century ago and has since grown into a thriving metropolis within the data processing landscape, so it is hardly new. However, it is new to us. This is a place where we will find we'll be able to use these automated unit testing techniques as freely and comfortably as we have with our current comfort level of writing programs using the ABAP language.

Why We Are Going There

We are going there primarily for *four* reasons:

1. Eliminate the drudgery associated with preparing and running manual tests.

   Most of us regard manually writing and running unit tests to be a process filled with dread and agony as we jump out of our comfort zone of programming and into the twilight zone of using spreadsheets and text editors to prepare a document to describe a unit test script. After that horrible experience, we then sit at a computer as we swap back and forth between the session presenting the unit test script and the session executing the software to be tested in a dizzying effort to execute in the software test session the instructions we are reading in the unit test script session. Such testing endeavors, where swapping between sessions is frequent, challenge our ability to pay attention to where we are and what we are intending to do with each swap. It is even worse if we are expected to update the unit test script with results as we step through it.

2. Shift the relatively long time it takes to manually run unit tests to more productive development pursuits.

   Hardly anyone would suggest that manually running a unit test script could be completed anywhere near as rapidly as an automated unit test could complete when one considers that an automated unit test typically runs to completion in a mere fraction of a second. If we were to aggregate all the time we have spent manually running unit tests over our careers, we might find that we have the time to complete all the component refactoring we had been unable to address, attend to all the technical debt that had accumulated over the years, and apply all the performance optimizations to those software components that have begun running more slowly in production and still have time left over for learning some new software feature or sharpening the skills we already possess.

3. Reap the benefits automated unit testing has on software design.

   In many cases, the very attempt to write an automated unit test for a component will reveal any weaknesses in the design of the production software. The inability to find a way to automatically test a component is a smell that should suggest there is a better way to design the software such that it is capable of being automatically tested. Accordingly, the production software design is improved when it is refactored to enable a passing automated unit test to be written for it. Code having such tests often gets better with each change, whereas code without such tests often gets worse with each change.

4. Instill confidence applying changes during maintenance.

   Statistics show that the initial development effort of writing a computer program consumes only a small fraction of the total time spent during its life cycle and that most of the time we devote to programming is in pursuit of maintenance efforts – change.[1]

[1]Software maintenance costs can be 75% of software total ownership costs. https://galorath.com/software-maintenance-costs/

> A significant reason offered by many experts for using an automated unit testing facility is that the tests run so rapidly it encourages the developer to run them frequently. Since automated unit tests can be run at the push of a button and often will complete faster than the time it takes to reach for and push that button, running automated unit tests after applying a maintenance change instantly instills in the developer the confidence that the most recent changes applied have introduced no new bugs.

How We Are Going to Get There

We are going to start where we are most comfortable and familiar and then move slowly and methodically until we have mastered the fundamentals of automated unit testing. This means we shall start from the familiar surroundings of manual unit testing (MUT) as practiced in our hometown of Mutville and travel along the path of least resistance to the automated unit testing (AUT) as practiced at our destination of Autropolis.

Along the way from Mutville to Autropolis, we will pass through the following districts:

- Software Quality
- xUnit
- ABAP Unit
- Rudiments
- Design for Testability
- Test Doubles
- Service Locator
- Leveraging the Service Locator
- Test-Driven Development
- Configurable Test Doubles
- Cultivating Good Test Writing Skills

Each district will present its own unique landscape distinguishing it from the other districts. Although we will use this book primarily to provide the directions for navigating the new terrain, we also will take the opportunity to pause in each district long enough to become more familiar with the new concepts we will encounter by performing exercises designed to strengthen our grasp of the nuances and idiosyncrasies each district has to offer. In the same way that merely reading a book about swimming could not sufficiently prepare us for the experience of actually jumping into the water for the first time, merely reading this book without performing the accompanying exercises similarly would leave us less than sufficiently prepared for the experience of actually using what we will be learning.

The first district we will encounter along the road from Mutville to Autropolis is known as Software Quality, a region where we can learn about what it takes to build good-quality software and the methods through which the level of software quality can be assessed. This is first because it establishes the reasons why we subject software to testing, providing us with a solid foundation for our trek through the remaining districts. This district is covered in Chapter 3.

Once we've learned about the things the Software Quality district has to offer, we'll proceed on to the district known as xUnit, a territory where the residents have transformed the art of unit testing by inventing a way to automate this process. Since it was first established about 30 years ago, it has grown over that time to have a significant impact on many of the languages used throughout the software industry. This district is traversed in Chapter 4.

In Chapter 5, we will explore ABAP Unit, a particular neighborhood of the xUnit district where the primary language spoken is ABAP. The local residents have found ways to adapt the peculiarities of the ABAP language to the same laws, regulations, and customs underpinning the automated unit testing that made the xUnit district famous.

Upon departing ABAP Unit, we'll head for a place known as Rudiments, where the residents have established procedures for engaging in practical activities designed to strengthen our understanding of the basic concepts associated with automated unit testing. This district is explored in Chapter 6.

After leaving Rudiments, we will continue on our way until reaching the district known as Design for Testability, where the residents excel at reorganizing components in such a way that promotes cleaner code while at the same time maintaining the ability to utilize the automated unit testing techniques we learned in Rudiments. Chapter 7 will guide us through this sector.

Farther down the road, we will move through Test Doubles, where the residents have mastered the art of deception and illusion, masquerading as dummy objects, fake objects, mock objects, stubs, and spies. This district is featured in Chapter 8.

After Test Doubles, we will cross into Service Locator, a place where the residents have organized their shared municipal services (animal control, fire alarm certification, tax reassessment, etc.) in such a way that all it takes is a call to the services distributor to arrange for the requested service. In most cases, the requester of a service is oblivious of the entity providing the service. Chapter 9 takes us through this district.

Beyond Service Locator lies Leveraging the Service Locator, where the residents have instituted a process for establishing control over entities providing services to other entities, enabling a service to be provided by the most appropriate entity based on the circumstances of the requester. For instance, the town dog catcher can respond to a resident calling animal control about a stray dog, but a call about a prowling cougar in a neighborhood might elicit a response from both the local police department and the Division of Wildlife Resources to tranquilize and safely move the animal to a remote area. We'll be escorted through this district by Chapter 10.

Next, we will traverse through the TDD district, where we will explore a process known as Test-Driven Development, the skill for which is in abundant supply among the district residents, all of whom are familiar with and adherents of a process whereby an automated test is written even before writing the corresponding production code it is intended to test – effectively putting the cart (test) before the horse (production code). It will be Chapter 11 that leads us through this district.

We will then traverse through Configurable Test Doubles, where the residents have found ways to eliminate the need to write their own explicit test double classes and instead rely on a software framework capable of simulating the presence of actual test doubles. This district is covered in Chapter 12.

Chapter 13 will guide us through the next district, a place known as Obtaining Code Coverage Information. The residents here have developed ways to determine the extent of the code covered by unit tests, enabling them to identify those parts of programs that remain without unit tests.

Next, we will pass through a place known for Cultivating Good Test Writing Skills. The folks who live here have learned many lessons about what makes for good unit tests and are willing to share this wisdom with visitors on their way to Autropolis. This district is covered in Chapter 14.

In Chapter 15, we finally will have reached our destination of Autropolis, where all the things we've learned along the way will enable us to walk the walk and talk the talk with the residents who have contributed to raising automated unit testing into a high art form.

Legacy Code

Our journey to Autropolis will consist of many encounters with ABAP code. Since the year 2000, there have been two different programming paradigms available for writing ABAP: procedural and object-oriented. Perhaps it is because the bulk of customized programs at a site had been written before the object-oriented flavor of ABAP became available that much of the ABAP code we see today is procedural, referred to euphemistically as *legacy code*.

Some testing scholars point out that legacy procedural code presents a more formidable challenge to writing unit tests than would be found with object-oriented code:

> *Legacy code … often refers to code that's hard to work with, hard to test, and usually even hard to read.*[2]
>
> *Anyone who has tried to retrofit automated unit tests onto legacy software can testify to the difficulty this raises.*[3]

Gerard Meszaros identifies six kinds of tests and lists them in approximate ascending order of difficulty, with "non-object-oriented legacy software" identified as most difficult:

> *As we move down the list, the software becomes increasingly more challenging to test. The irony is that many teams "get their feet wet" by trying to retrofit tests onto an existing application. … Unfortunately, many teams fail to test the legacy software successfully, which may then prejudice them against trying automated testing …*[4]

[2]Osherove, Roy, *The Art of Unit Testing*, second edition, Manning, 2014, p. 9

[3]Meszaros, Gerard, *xUnit Test Patterns: Refactoring Test Code*, Addison Wesley, 2007, p. 40

[4]Ibid, p. 176

These quotations may seem alarming, but we will find that all things are relative.[5] My reason for including them is to set the context for how we are going to get to Autropolis. Although it may be much easier to write automated unit tests for object-oriented ABAP, it is far more likely that (1) the bulk of the code at your site is legacy procedural code and (2) there remain many ABAP programmers who have not yet become comfortable with the object-oriented paradigm, so even their new code still is being written in a procedural style. Accordingly, to appeal to the widest cross-section of ABAP programmers, the procedural programming paradigm is used intentionally for the ABAP code examples in this book and the accompanying exercises. Programmers already writing object-oriented code should have no problem adapting the concepts presented in this book to their everyday activities, while programmers unfamiliar with object-oriented programming will not find themselves trudging through code difficult to understand.

Calisthenics

Along the way from Mutville to Autropolis, we will pause occasionally to perform some calisthenics by completing a few of the more than 180 exercises associated with this book. Each exercise presents or reinforces some concept presented in this book by introducing minor code changes into an ABAP program and then executing both its production path and unit test path to observe the effects of those changes. Indeed, some exercises will introduce changes to the program that will become reversed or discarded by a subsequent exercise so that we can more fully understand why some seemingly appropriate implementation techniques should be avoided.

It will take some time, effort, and determination to complete all of the exercises, but the end result is to become comfortable with the techniques used for writing unit tests as well as to understand how a production program can be refactored to enable automated unit testing upon it. As you make your way through the exercises, you may find yourself questioning how a particular exercise is intended to help you understand anything associated with automated unit testing since its practical benefit is not obvious. Perhaps the following anecdote might help.

In the movie *The Karate Kid*, Mister Miyagi asks adolescent Daniel to agree to a pact: in order to learn karate, Daniel is to spend each training day at Miyagi's house

[5]A phrase often attributed to Albert Einstein

performing whatever physical training Miyagi asks of him, without question. Daniel agrees, and on the first day, Miyagi shows Daniel the technique he wants used to apply wax to his cars, which Miyagi describes using the phrases "wax on" and "wax off." Daniel complies and is exhausted after spending the day waxing all of Miyagi's many cars.

The next day Miyagi shows Daniel how to "sand the floor," and Daniel spends the day using this technique on all the wooden decks at Miyagi's house. The next day Miyagi introduces Daniel to "paint the fence" and the following day to "paint the house." With each day, Daniel grows more and more frustrated that he is not learning karate, as promised, but is simply spending his time making home improvements to Miyagi's house. Daniel confronts Miyagi, complaining bitterly about all the work he has completed yet still knowing nothing about karate. In a scene exquisitely capturing the essence of "learn by doing," Miyagi stands facing Daniel and asks him to show the motions for "wax on" and "wax off," which Daniel does as Miyagi simultaneously performs the corresponding offensive karate maneuver for which Daniel's motions are the defense. Miyagi continues, having Daniel show the motions for "sand the floor," followed by "paint the fence" and "paint the house," each time attacking Daniel with the corresponding offensive karate maneuver. Afterward, Miyagi bows to Daniel and walks away in silence, leaving Daniel dumbfounded by suddenly realizing how much he has learned about karate without even knowing he was learning it.

Similarly, it is recommended that you simply perform the exercises even though at the time you may not grasp the benefit of having done so. It may not improve your karate skills, though this is only speculation, but certainly it should reinforce and solidify your comprehension of the concepts associated with automated unit testing.

Summary

The road map for traveling from Mutville to Autropolis was presented, specifically that the reader will be traveling along a path leading from the familiar surroundings of their hometown of Mutville, a place where arduous and time-consuming manual unit testing (MUT) still reigns, to the less familiar but highly mechanized town of Autropolis, a place where automated unit testing (AUT) has displaced outdated manual efforts of testing. It has oriented and prepared the reader for the journey, explaining where we are, where we are going, why we are going there, and how we are going to get there. Primarily it will be procedural code used with the examples throughout the book and with the accompanying exercise programs in order to appeal to the largest cross-section of