1. Record Nr. UNINA9910484705403321

Autore	Hardy Paul David
Titolo	Improving the quality of ABAP code : striving for perfection / / Paul David Hardy
Pubbl/distr/stampa	[Place of publication not identified] : , : Apress, , [2021] ©2021
ISBN	1-4842-6711-7
Descrizione fisica	1 online resource (526 pages)
Disciplina	005.133
Soggetti	ABAP/4 (Computer program language)
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Nota di contenuto	Intro -- Table of Contents -- About the Author -- About the Technical Reviewer -- Acknowledgments -- Introduction -- Chapter 1: Why Object-Oriented Programming Is a Must for Code Quality -- Why OO Has Never Taken Off in ABAP World -- OO Benefits: The Theory -- Describing OO Programs: UML -- Describing OO Programs: BON -- Seamlessness/Reversibility -- Seamlessness via an Automated Tool -- Seamlessness via Naming Conventions -- Design by Contract -- Design by Unit Tests -- Designing a Worldwide OO Program -- What Is the Existing System State? -- Why Is the Existing System State Not Portable? -- What Can Be Done About Fixing That Problem? -- Breaking Dependencies in General -- Breaking Dependencies via Packages -- Breaking Dependencies via Interfaces -- What Is an Interface in OO Terms? -- How SAP Uses Interfaces to Manage User Exits -- Not Breaking the System -- My Journey to OO Happiness -- OO Happiness: The Theory -- The Clean Coder -- Clean Code -- Head-First Design Patterns -- More on Design Patterns -- OO Happiness: The Reality -- Writing an Interactive Executable Report in OO -- Writing a DYNPRO Program in OO -- Experiment -- Rewriting a Huge, Business-Critical Program in OO -- Slowly Transforming a Huge, Business-Critical Program to OO -- OO Benefits: The Reality -- Soft Benefits -- Design Thinking -- Ease of Maintenance -- No One Goes Back -- Hard Benefits -- Avoidance of Syntax Errors -- Parameter Handling -- IMPORTING -- EXPORTING -- TABLES -- RETURNING -- OPTIONAL -- NAMES -- TYPES

-- FORMULAS -- Reuse -- Making Code Testable -- Conclusion -- Recommended Reading -- Articles -- Books -- Chapter 2: Why Test-Driven Development Is a Must for Code Quality -- TDD Theory -- Fragile Code -- Legacy Code -- Automated Regression Tests -- Dependencies and How to Break Them -- The TDD Development Cycle: RED/GREEN/BLUE -- My Journey to TDD Happiness.
Testing After the Event -- Testing Before the Event -- TDD Workflow in Eclipse -- Creating Test Doubles -- Creating a New Test Method -- Coding the Test Method -- Coding the THEN Method -- Coding the GIVEN Method -- Coding the WHEN Method -- Writing the Production Code -- The BLUE Phase -- Does This Actually Give You a Benefit? -- From a Gut Feeling . . . -- . . . to a Concrete Example -- Some Unit Tests Might Seem Pointless . . . -- It's Too Short to Test -- It's Too Simple to Test -- . . . but They Are Not! -- Message in a Model -- You Can't Get There from Here -- Simpler but Wrong -- Why TDD Has Never Taken Off -- I Already Have an Automated Testing Framework -- My Program Is Too Complicated for TDD -- My Program Is Too Simple for TDD -- TDD Is Far Too Expensive -- TDD Reduces Development Costs -- TDD Reduces the Cost of Fixing Bugs -- TDD Reduces the Financial Risk of Failure -- Conclusion -- Recommended Reading -- Chapter 3: Clarity: The First Pillar of Code Quality -- Refactoring: Automated and Manual Checks -- Automated Checks -- Syntax Check -- Extended Program Check -- Code Inspector -- ABAP Open Checks -- Code Pal -- Remote ATC Checks -- Continuous Integration -- Manual Checks -- Clean ABAP -- ABAP Gore -- Creating a Personalized Checklist -- Code Complexity -- Huge Routines -- Confusing Code -- Double Negatives -- Text Symbols -- Pointless Variables -- Contradictory Instructions -- END-OF-SELECTION -- DATA Declarations -- Unrelated Tasks -- Not Being Able to Locate a Routine -- Duplicate Code -- Global Variables -- Why Are Global Variables Bad? -- What Can You Do About Global Variables? -- Global Variables Versus Member Variables -- Naming -- Method/Routine Naming -- Misleading Names -- Totally Incorrect Names -- Sloppy Naming -- Parameter Naming -- Named Parameters -- Functional Methods -- Variable Naming -- Hungarian Notation.
Prefixes in an OO Context -- Random Naming Conventions -- Misleading Names -- Very Old Programs -- Inline Declarations -- Self-Documenting -- German Acronyms -- Magic Numbers -- Hard Coding -- Constants -- Redundant Constants -- Meaningless Constants -- How Not to Use Constants -- ABAP Data Dictionary Object Naming -- Tables -- Transaction Codes -- Structures -- Indexes -- Program Names -- CDS Views -- How Correct Naming Enables Reuse -- Comments -- Why, Not How -- Quotation Marks Versus Asterisks -- Meaningless Comments -- Incorrect Comments -- Reference Numbers in Comments -- Comments Going for a Walk -- Commented-Out Code -- It Makes the Program Harder to Follow -- It Causes Short Dumps -- It Doesn't Always Work -- It Should Be Deleted -- Documentation -- Conclusion -- Recommended Reading -- Chapter 4: Stability: The Second Pillar of Code Quality -- Principle of Least Astonishment -- Enhancement Category -- Material Substitution -- Data Declarations in Modules -- Hashed Tables -- Incorrectly Typed RETURNING Parameter -- Programming by Accident -- Wrong Code That Works -- Incorrect Behavior Being Viewed as Correct -- Archaic ABAP Statements -- Strange Data Declarations -- Implicit Work Area -- What Do You Do? -- Don't Repeat Yourself -- Time/Difficulty -- Before the Event -- After the Event -- Riskiness -- Surgeon Example -- Drill-Down Example -- Text Names Example -- Payer Example -- Avoiding Repetition in OO Programming -- Other Common Causes of Instability -- Global

Variables -- Cannon Example -- Half-Dog Half-Cat Example -- Function Modules -- Cross-Program Calls -- Table-Based Work Areas -- Parameters -- Importing Parameters -- Exporting Parameters -- Fully Typed Parameters -- Big Trouble with Big Signatures -- Memory Problems -- Dealing with Instability: Using Code -- Problems That Virtually Always Happen -- Fuzzy Searches -- Spreadsheets. Problems That Are Likely to Happen -- SY-SUBRC -- BAPIs -- BDCs -- Field Symbols -- Problems That Really Shouldn't Happen -- Self-Repair: Example 1 -- Self-Repair: Example 2 -- Problems That Should Never, Ever Happen -- Spotting the Impossible -- Dealing with the Impossible -- Dealing with Instability: Using Humans -- What's Wrong #1 -- What's Wrong #2 -- Conclusion -- Recommended Reading -- Chapter 5: Performance: The Third Pillar of Code Quality -- CPO Concept -- Daily Dumps -- Annual In-House CPO -- Static Checks -- Geometric Loops -- Secondary Indexes for Internal Tables -- Runtime Checks -- ST05 in General -- Identical SELECTS -- How to Spot the Problem -- Strategies to Deal with the Problem -- Real-Life Example -- Stuttering -- Asking Stupid Questions -- Reading More Data Than Needed -- Selecting More Columns Than You Need -- Selecting More Rows Than You Need -- The Behavior Never Made Sense -- The Behavior No Longer Makes Sense -- The Behavior Makes Sense -- Existence Checks -- Multiple Reads on the Same Table -- Contract Example -- Partner Function Example -- Shipment Cost Example -- Using FOR ALL ENTRIES -- Indexes -- Indexes: Always Using One -- Indexes: Missing the First Field -- Indexes: SKIP SCAN -- Nested SELECTs -- "Bad" Joins on Database Tables -- Postmortem Checks -- Standard SAP Transactions for Troubleshooting Performance Problems -- SAT -- ST04 -- SRTCM -- Technical Attributes of Database Tables -- Indexes -- Pointless Index -- Terrible Index -- Really Good Index -- Buffering -- Possible Buffering Settings -- Example of How Buffering Can Help -- Common Misconception About Generic Buffering -- How to Decide Which Z Tables to Buffer -- Postmortem Tricks -- Batch Jobs -- Deadlocks -- Sneaky Tricks -- Database Reads in a Loop -- Using Standard SAP "Buffering" Modules -- When to Use Them -- KNA1_SINGLE_READER -- Prefilling Buffers. Constants -- INTO CORRESPONDING -- DDIC Information -- Using Standard SAP Functions Incorrectly -- Conclusion -- Chapter 6: User Friendliness: Ensuring UI Quality -- General Philosophy -- Difference Between UI and UX -- Don't Make Me Think -- Waterfall Projects -- Consistency -- Standards -- Applying Industry Standards -- Non-Standard Icon Appearance -- Non-Standard Icon Usage -- Inconsistency -- Inconsistency in F4 Helps -- Inconsistency in Pop-Ups -- Inconsistency in Master Data Transactions -- Ease of Use -- Laying Traps for the User -- Hiding Icons for No Reason -- Hiding Fields for No Reason -- Slightly Hidden Field -- Totally Hidden Field -- Incorrect Use of Check Boxes -- Confusing the User -- Giving Incorrect Options -- Giving No Options at All -- Accessibility -- Explaining Things to the User -- Avoiding Abbreviations -- Custom Data Elements -- Custom F1 Help -- Custom Domains -- Error Prevention -- Inviting Errors -- Self-Destruct Button -- Stopping Dumps Before They Begin -- Self-Service -- Logging -- Linking Errors to Training Material -- Sneaky Trick: ALV Filter Not Working -- Error Handling -- Shouting at the User -- Preventing the User from Fixing the Problem -- Making the User Reenter Data -- Making It Impossible to Fix the Problem -- Enabling the User to Fix the Problem -- Mandatory Fields -- Taking the User to the Source of the Problem -- Example: IDoc Application Log -- Documentation -- Documentation Guidelines -- Documentation Terminology -- Documentation for Dialog Transactions --

Documentation for Developers -- Conclusion -- Recommended Reading -- Chapter 7: User Exits: Defusing a Potential Time Bomb -- User Exits in On-Premises SAP Systems -- VOFM Routines -- Repairs -- FORM-Based User Exits -- CMOD User Exits -- BAdi User Exits -- The Enhancement Framework -- User Exits in Your Own Z Programs -- User Exits in Cloud SAP Systems.
UI Extensibility.