1. Record Nr. UNINA9910824829703321

Autore	Dingle Adair
Titolo	Software essentials : design and construction / / Adair Dingle, Seattle University, Washington, USA
Pubbl/distr/stampa	Boca Raton : , : Taylor & Francis, , [2014] ©2014
ISBN	0-429-06344-X 1-4398-4120-9
Edizione	[1st edition]
Descrizione fisica	1 online resource (432 p.)
Collana	Chapman & Hall/CRC Innovations in Software Engineering and Software Development
Classificazione	COM051230COM051300
Disciplina	005.1/2
Soggetti	Software architecture Computer software - Development
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Note generali	A Chapman and Hall book.
Nota di bibliografia	Includes bibliographical references.
Nota di contenuto	Front Cover; Contents; Preface; Acknowledgments; Detailed Book Outline; Chapter 1: Software Complexity and Modeling; Chapter 2: Software Development; Chapter 3: Functionality; Chapter 4: Memory; Chapter 5: Design and Documentation; Chapter 6: Structural Design; Chapter 7: Behavioral Design; Chapter 8: Design Alternatives and Perspectives; Chapter 9: Software Correctness; Chapter 10: Software Longevity; Glossary: Definitions and Conceptual Details; References; Appendix A: Memory and the Pointer Construct; Appendix B: Heap Memory and Aliases; Appendix C:Function Pointers Appendix D: Operator OverloadingBack Cover
Sommario/riassunto	Preface Why this book? Why should you read this book? The short answer is to study software design from a structured but hands-on perspective and to understand different models of control flow, memory, dynamic behavior, extensibility, et cetera Software complexity and the growing impact of legacy systems motivate a renewed interest in software design and modeling. We emphasize design (and construction) in this text, using and contrasting C# and C++. Many CS texts are 'learn to' books that focus on one programming language or tool. When perspective is so limited to a specific tool or programming

language, high-level concepts are often slighted. Students may gain exposure to an idea via a 'cookbook' implementation and thus fail to truly absorb essential concepts. Students and/or practitioners can understand and apply design principles more readily when such concepts are explicitly defined and illustrated. Design, not just syntax, must be stressed. The progression of programming languages, software process methodologies and development tools continues to support abstraction: software developers should exploit this abstraction and solve problems (design) without being tied to a particular syntax or tool. Software design and modeling are neither new nor trendy topics. Software development often focuses on immediate effect: implement, test (minimally) and deploy. Yet, the complexity, scale and longevity of modern software require an intricate understanding of a software system as a whole -- components and relationships, user interfaces, persistent data, et cetera To accommodate existing use while preserving longevity, a software developer must look forward for extensibility and backward for compatibility. Hence, software developers must understand software design. --