a discipline of programming

a discipline of programming represents a structured approach to software development that emphasizes best practices, methodologies, and principles to create efficient, reliable, and maintainable code. This discipline encompasses various programming paradigms, techniques, and tools that guide developers in writing high-quality software. Understanding the core aspects of a discipline of programming is essential for both novice and experienced programmers aiming to enhance their skills and deliver robust applications. The discipline also addresses common challenges such as debugging, testing, and optimizing code, thereby improving overall project outcomes. This article explores the fundamental components of a discipline of programming, its methodologies, practical applications, and the benefits it brings to software development teams. The following sections provide an in-depth look at key elements that define and support this essential programming discipline.

Understanding the Fundamentals of a Discipline of Programming

A discipline of programming is centered on a systematic approach to writing software that adheres to defined standards and principles. It involves understanding programming languages, algorithms, data structures, and software design patterns. At its core, this discipline aims to produce code that is not only functional but also readable, maintainable, and scalable. The fundamentals include mastering syntax, semantics, and the logic that drives program execution.

Programming Paradigms

Various programming paradigms form the basis of a discipline of programming, each offering distinct ways to conceptualize and solve problems. The most common paradigms include procedural, object-oriented, functional, and declarative programming. Each paradigm influences how developers structure their code and manage program state, contributing to the overall effectiveness of the software.

Importance of Algorithms and Data Structures

Algorithms and data structures are essential components within the discipline of programming. Efficient algorithms optimize performance, while appropriate data structures organize data effectively. Together, they ensure that software applications run smoothly and handle complex tasks with minimal resource consumption.

Software Design Principles

Adhering to software design principles such as SOLID, DRY (Don't Repeat Yourself), and KISS (Keep It Simple, Stupid) is crucial in a discipline of programming. These principles help developers create modular, reusable, and understandable codebases, reducing technical debt and facilitating easier maintenance and upgrades.

Key Methodologies in a Discipline of Programming

Methodologies provide structured processes and frameworks that programmers follow to manage software development effectively. These approaches guide the planning, coding, testing, and deployment stages, aligning team efforts and improving project outcomes.

Agile Development

Agile methodology focuses on iterative development, where requirements and solutions evolve through collaboration between cross-functional teams. This approach supports adaptability and continuous improvement, making it a staple in modern a discipline of programming.

Test-Driven Development (TDD)

TDD is a methodology where developers write tests before writing the corresponding code. This practice ensures that code meets requirements from the outset and helps catch defects early, enhancing code reliability and maintainability.

Version Control Systems

Utilizing version control systems such as Git is a fundamental part of a discipline of programming. These tools enable developers to track changes, collaborate efficiently, and manage code histories, which are vital for team-based software projects.

Practical Applications and Best Practices

Applying the principles and methodologies of a discipline of programming in real-world projects leads to higher quality software products. Best practices serve as guidelines that minimize errors and optimize development workflows.

Code Reviews and Pair Programming

Code reviews involve systematic examination of code by peers to identify issues and improve quality. Pair programming, where two developers work together at one workstation, fosters knowledge sharing and immediate feedback, both reinforcing disciplined programming habits.

Documentation and Commenting

Comprehensive documentation and clear commenting are integral to a discipline of programming. They provide context and explanations for code segments, aiding future developers in understanding and maintaining the codebase effectively.

Continuous Integration and Continuous Deployment (CI/CD)

CI/CD pipelines automate the process of integrating code changes, running tests, and deploying applications. This automation ensures that software is consistently tested and delivered, reducing manual errors and accelerating release cycles.

Challenges and Solutions within a Discipline of Programming

Despite its structured nature, a discipline of programming faces challenges such as complexity management, debugging difficulties, and keeping up with rapidly evolving technologies. Addressing these challenges is key to maintaining productivity and code quality.

Managing Code Complexity

As software projects grow, codebases can become complex and hard to manage. Techniques like modular programming, refactoring, and adhering to design patterns help control complexity and improve code clarity.

Effective Debugging Techniques

Debugging is a critical aspect of a discipline of programming. Employing systematic debugging methods, using specialized tools, and writing testable code contribute to quicker identification and resolution of defects.

Keeping Skills Updated

The fast pace of technological advancements requires programmers to continually update their knowledge. Participating in training, reading technical literature, and engaging with programming communities supports ongoing professional development.

Future Trends in a Discipline of Programming

The discipline of programming is continuously evolving with innovations in artificial intelligence, automation, and development tools. These trends are shaping how developers write code and manage software projects.

Artificial Intelligence and Machine Learning Integration

AI and machine learning are increasingly integrated into programming tools, offering intelligent code suggestions, automated testing, and bug detection. This integration enhances productivity and code quality within the discipline of programming.

Low-Code and No-Code Platforms

Low-code and no-code development platforms are democratizing software creation by enabling users with limited programming knowledge to build applications. While these platforms simplify development, understanding core programming disciplines remains essential for complex projects.

Emphasis on Cybersecurity

As cyber threats become more sophisticated, incorporating security considerations into every stage of programming is a growing focus. Secure coding practices and vulnerability assessments are critical components of a discipline of programming moving forward.

- Adopt structured programming methodologies to enhance code quality.
- Leverage modern tools like version control and CI/CD pipelines for efficient workflows.
- Focus on continuous learning to keep pace with emerging technologies.
- Implement thorough testing and debugging techniques to ensure reliability.
- Emphasize documentation and collaboration to maintain sustainable codebases.

Frequently Asked Questions

What is meant by 'a discipline of programming'?

A discipline of programming refers to the systematic approach and set of principles, methodologies, and practices used to write, test, and maintain software in a structured and efficient manner.

Why is discipline important in programming?

Discipline in programming ensures code quality, maintainability, and reduces errors by encouraging consistent coding standards, thorough testing, and proper documentation.

What are some common practices in the discipline of programming?

Common practices include writing clean code, using version control, performing code reviews, following design patterns, testing thoroughly, and continuous learning.

How does disciplined programming impact software development teams?

Disciplined programming fosters better collaboration, reduces technical debt, improves project predictability, and facilitates easier onboarding of new team members.

What role does discipline play in debugging and problem-solving?

Discipline helps programmers approach debugging methodically, using systematic techniques such as reproducing bugs, isolating causes, and documenting fixes to efficiently resolve issues.

Can discipline in programming improve career prospects for developers?

Yes, disciplined programmers are often more reliable, produce higher-quality code, and are valued in the industry, which can lead to better job opportunities and career growth.

Additional Resources

1. *Clean Code: A Handbook of Agile Software Craftsmanship*
This book by Robert C. Martin emphasizes writing clean, readable, and maintainable code. It offers practical advice on improving code quality through best practices, refactoring, and understanding code smells. Developers learn how to create software that is easier to understand and modify, which ultimately leads to better collaboration and fewer bugs.

2. *Introduction to Algorithms*
Often referred to as "CLRS" after its authors Cormen, Leiserson, Rivest, and Stein, this comprehensive text covers a broad range of algorithms in depth. It provides detailed explanations of algorithm design and analysis, including sorting, searching, dynamic programming, and graph algorithms. This book is a foundational resource for computer science students and professionals alike.

3. *Design Patterns: Elements of Reusable Object-Oriented Software*
Written by the "Gang of Four," this classic book introduces 23 design patterns that solve common software design problems. It helps programmers understand how to build flexible and reusable object-oriented systems. The book covers creational, structural, and behavioral patterns with examples

primarily in C++ and Smalltalk.

4. *JavaScript: The Good Parts*
Douglas Crockford focuses on the core features of JavaScript that make it a powerful and expressive language. The book distills the language down to its most reliable and robust components, avoiding common pitfalls and bad practices. It is essential reading for developers aiming to write high-quality JavaScript code.

5. *Effective Python: 90 Specific Ways to Write Better Python*
Brett Slatkin provides actionable advice and best practices for writing clean, idiomatic Python code. The book covers topics such as data structures, functions, concurrency, and testing, with a focus on improving performance and maintainability. It is well-suited for Python developers looking to deepen their understanding of the language.

6. *Refactoring: Improving the Design of Existing Code*
Martin Fowler's seminal work guides developers through the process of restructuring existing code without changing its external behavior. The book introduces patterns for identifying code smells and applying refactorings to make code more understandable and extensible. It is an invaluable resource for maintaining and evolving legacy codebases.

7. *Programming Rust: Fast, Safe Systems Development*
This book covers Rust, a systems programming language focused on safety and performance. It explains Rust's ownership model, concurrency mechanisms, and type system, enabling developers to build fast and reliable software. The book is ideal for those interested in system-level programming with modern tools.

8. *Head First Design Patterns*
Using a visually rich format, this book introduces design patterns in an engaging and accessible way. It focuses on practical examples and real-world scenarios to help readers grasp complex concepts intuitively. A great resource for beginners and those new to object-oriented design.

9. *The Pragmatic Programmer: Your Journey to Mastery*
Andrew Hunt and David Thomas share practical tips and philosophies for becoming a versatile and effective programmer. The book covers a wide range of topics including code craftsmanship, debugging, and career development. It encourages continuous learning and proactive problem-solving in software development.

A Discipline Of Programming

a discipline of programming: A Discipline of Programming Edsger W. Dijkstra, 1976 Executional abstraction; The role of programming languages; States and their characterization; The characterization of semantics; The semantic characterization of a programming language; Two

theorems; On the design of properly terminating; Euclid's algorithm revisited; The formal treatment of some small examples; The linear search theorem; The problem of the next permutation.

a discipline of programming: A discipline of programming Edsger W. Dijkstra, 1976

a discipline of programming: A Discipline of Programming Edsger W. Dijkstra, 1976 NULL

a discipline of programming: A Discipline of programming Edsger Wybe Dykstra, 1976

a discipline of programming: Mathematics of Program Construction Tarmo Uustalu, 2006-06-27 This volume contains the proceedings of the 8th International Conference on Mathematics of ProgramConstruction, MPC 2006,held at Kuressaare, Estonia, July 3-5, 2006, colocated with the 11th International Conference on Algebraic Methodology and Software Technology, AMAST 2006, July 5-8, 2006.
TheMPCconferencesaimtopromotethedevelopmentofmathematicalpr- ciples and techniques that are demonstrably useful and usable in the process of constructing computer programs. Topics of interest range from algorithmics to support for program construction in programming languages and systems. The previous MPCs were held at Twente, The Netherlands (1989, LNCS 375), Oxford, UK (1992, LNCS 669), Kloster Irsee, Germany (1995,LNCS 947), Marstrand, Sweden (1998, LNCS 1422), Ponte de Lima, Portugal (2000, LNCS 1837), Dagstuhl, Germany (2002, LNCS 2386) and Stirling, UK (2004, LNCS 3125, colocated with AMAST 2004). MPC 2006 received 45 submissions. Each submission was reviewed by four Programme Committee members or additional referees. The committee decided to accept 22 papers. In addition, the programme included three invited talks by Robin Cockett (University of Calgary, Canada), Olivier Danvy (Aarhus Univ- sitet, Denmark) and Oege de Moor (University of Oxford, UK). The review process and compilation of the proceedings were greatly helped by Andrei Voronkov's EasyChair system that I can only recommend to every programme chair. MPC 2006 had one satellite workshop, the Workshop on Mathematically Structured Functional Programming, MSFP 2006, organized as a small wo- shop of the FP6 IST coordination action TYPES. This took place July 2, 2006.

a discipline of programming: *Mathematics of Program Construction* Bernhard Möller, 1995-07-10 This volume constitutes the proceedings of the Third International Conference on the Mathematics of Program Construction, held at Kloster Irsee, Germany in July 1995. Besides five invited lectures by distinguished researchers there are presented 19 full revised papers selected from a total of 58 submissions. The general theme is the use of crisp, clear mathematics in the discovery and design of algorithms and in the development of corresponding software and hardware; among the topics addressed are program transformation, program analysis, program verification, as well as convincing case studies.

a discipline of programming: *Mathematics of Program Construction* Claude Bolduc, Jules Desharnais, Bechir Ktari, 2010-06 This book constitutes the refereed proceedings of the 10th International Conference on Mathematics of Program Construction, MPC 2010, held in Québec City, Canada in June 2010. The 19 revised full papers presented together with 1 invited talk and the abstracts of 2 invited talks were carefully reviewed and selected from 37 submissions. The focus is on techniques that combine precision with conciseness, enabling programs to be constructed by formal calculation. Within this theme, the scope of the series is very diverse, including programming methodology, program specification and transformation, program analysis, programming paradigms, programming calculi, programming language semantics, security and program logics.

a discipline of programming: *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming* Ricardo Queirós, 2014-11-30 Courses in computer programming combine a number of different concepts, from general problem-solving to mathematical precepts such as algorithms and computational intelligence. Due to the complex nature of computer science education, teaching the novice programmer can be a challenge. Innovative Teaching Strategies and New Learning Paradigms in Computer Programming brings together pedagogical and technological methods to address the recent challenges that have developed in computer programming courses. Focusing on educational tools, computer science concepts, and educational design, this book is an essential reference source for teachers, practitioners, and scholars interested in improving the

success rate of students.

a discipline of programming: FM 2014: Formal Methods Cliff Jones, Pekka Pihlajasaari, Jun Sun, 2014-04-18 This book constitutes the refereed proceedings of the 19th International Symposium on Formal Methods, FM 2014, held in Singapore, May 2014. The 45 papers presented together with 3 invited talks were carefully reviewed and selected from 150 submissions. The focus of the papers is on the following topics: Interdisciplinary Formal Methods, Practical Applications of Formal Methods in Industrial and Research Settings, Experimental Validation of Tools and Methods as well as Construction and Evolution of Formal Methods Tools.

a discipline of programming: Mathematics of Program Construction Jan L.A. van de Snepscheut, 1989-06-07 The papers included in this volume were presented at the Conference on Mathematics of Program Construction held from June 26 to 30, 1989. The conference was organized by the Department of Computing Science, Groningen University, The Netherlands, at the occasion of the University's 375th anniversary. The creative inspiration of the modern computer has led to the development of new mathematics, the mathematics of program construction. Initially concerned with the posterior verification of computer programs, the mathematics have now matured to the point where they are actively being used for the discovery of elegant solutions to new programming problems. Initially concerned specifically with imperative programming, the application of mathematical methodologies is now established as an essential part of all programming paradigms - functional, logic and object-oriented programming, modularity and type structure etc. Initially concerned with software only, the mathematics are also finding fruit in hardware design so that the traditional boundaries between the two disciplines have become blurred. The varieties of mathematics of program construction are wide-ranging. They include calculi for the specification of sequential and concurrent programs, program transformation and analysis methodologies, and formal inference systems for the construction and analysis of programs. The mathematics of specification, implementation and analysis have become indispensable tools for practical programming.

a discipline of programming: Unifying Theories of Programming Shengchao Qin, 2010 Based on the pioneering work of C.A.R.

a discipline of programming: The Software Life Cycle Darrel Ince, Derek Andrews, 2014-05-20 The Software Life Cycle deals with the software lifecycle, that is, what exactly happens when software is developed. Topics covered include aspects of software engineering, structured techniques of software development, and software project management. The use of mathematics to design and develop computer systems is also discussed. This book is comprised of 20 chapters divided into four sections and begins with an overview of software engineering and software development, paying particular attention to the birth of software engineering and the introduction of formal methods of software development. The next section explores some aspects of software engineering that tend to get ignored in the literature, including functional programming, functional-programming languages, and relational databases. The reader is then introduced to structured methods of software development, along with software project management. The final chapter is devoted to software testing, which can be functional or nonfunctional. This monograph will be useful to software engineers and designers.

a discipline of programming: Edsger Wybe Dijkstra Krzysztof R. Apt, Tony Hoare, 2022-07-14 Edsger Wybe Dijkstra (1930–2002) was one of the most influential researchers in the history of computer science, making fundamental contributions to both the theory and practice of computing. Early in his career, he proposed the single-source shortest path algorithm, now commonly referred to as Dijkstra’s algorithm. He wrote (with Jaap Zonneveld) the first ALGOL 60 compiler, and designed and implemented with his colleagues the influential THE operating system. Dijkstra invented the field of concurrent algorithms, with concepts such as mutual exclusion, deadlock detection, and synchronization. A prolific writer and forceful proponent of the concept of structured programming, he convincingly argued against the use of the Go To statement. In 1972 he was awarded the ACM Turing Award for “fundamental contributions to programming as a high,

intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design." Subsequently he invented the concept of self-stabilization relevant to fault-tolerant computing. He also devised an elegant language for nondeterministic programming and its weakest precondition semantics, featured in his influential 1976 book A Discipline of Programming in which he advocated the development of programs in concert with their correctness proofs. In the later stages of his life, he devoted much attention to the development and presentation of mathematical proofs, providing further support to his long-held view that the programming process should be viewed as a mathematical activity. In this unique new book, 31 computer scientists, including five recipients of the Turing Award, present and discuss Dijkstra's numerous contributions to computing science and assess their impact. Several authors knew Dijkstra as a friend, teacher, lecturer, or colleague. Their biographical essays and tributes provide a fascinating multi-author picture of Dijkstra, from the early days of his career up to the end of his life.

a discipline of programming: Into the Cloud David J. Emmick, 2009-09-28 This book integrates standard practices and operations thinking into design and architecture of large scale services which is something I am uniquely qualified for given a varied background from development to operations. This book is a presentation of innovative designs and programs for large scale services and a look into the future of service design and architecture.

a discipline of programming: The French School of Programming Bertrand Meyer, 2024-04-29 The French School of Programming is a collection of insightful discussions of programming and software engineering topics, by some of the most prestigious names of French computer science. The authors include several of the originators of such widely acclaimed inventions as abstract interpretation, the Caml, OCaml and Eiffel programming languages, the Coq proof assistant, agents and modern testing techniques. The book is divided into four parts: Software Engineering (A), Programming Language Mechanisms and Type Systems (B), Theory (C), and Language Design and Programming Methodology (D). They are preceded by a Foreword by Bertrand Meyer, the editor of the volume, a Preface by Jim Woodcock providing an outsider's appraisal of the French school's contribution, and an overview chapter by Gérard Berry, recalling his own intellectual journey. Chapter 2, by Marie-Claude Gaudel, presents a 30-year perspective on the evolution of testing starting with her own seminal work. In chapter 3, Michel Raynal covers distributed computing with an emphasis on simplicity. Chapter 4, by Jean-Marc Jézéquel, former director of IRISA, presents the evolution of modeling, from CASE tools to SLE and Machine Learning. Chapter 5, by Joëlle Coutaz, is a comprehensive review of the evolution of Human-Computer Interaction. In part B, chapter 6, by Jean-Pierre Briot, describes the sequence of abstractions that led to the concept of agent. Chapter 7, by Pierre-Louis Curien, is a personal account of a journey through fundamental concepts of semantics, syntax and types. In chapter 8, Thierry Coquand presents "some remarks on dependent type theory". Part C begins with Patrick Cousot's personal historical perspective on his well-known creation, abstract interpretation, in chapter 9. Chapter 10, by Jean-Jacques Lévy, is devoted to tracking redexes in the Lambda Calculus. The final chapter of that part, chapter 11 by Jean-Pierre Jouannaud, presents advances in rewriting systems, specifically the confluence of terminating rewriting computations. Part D contains two longer contributions. Chapter 12 is a review by Giuseppe Castagna of a broad range of programming topics relying on union, intersection and negation types. In the final chapter, Bertrand Meyer covers "ten choices in language design" for object-oriented programming, distinguishing between "right" and "wrong" resolutions of these issues and explaining the rationale behind Eiffel's decisions. This book will be of special interest to anyone with an interest in modern views of programming — on such topics as programming language design, the relationship between programming and type theory, object-oriented principles, distributed systems, testing techniques, rewriting systems, human-computer interaction, software verification… — and in the insights of a brilliant group of innovators in the field.

a discipline of programming: *On the Foundations of Computing* Giuseppe Primiero, 2020 On

The Foundations of Computing is a technical, historical and conceptual investigation in the three main methodological approaches to the computational sciences: mathematical, engineering and experimental. The first part of the volume explores the background behind the formal understanding of computing, originating at the end of the XIX century, and it invesitagtes the formal origins and conceptual development of the notions of computation, algorithm and program. The second part of the volume overviews the construction of physical devices to perform automated tasks and it considers associated technical and conceptual issues. We start from the design and construction of the first generation of computing machines, explore their evolution and progress in engineering (for both hardware and software), and investigate their theoretical and conceptual problems. The third part of the volume analyses the methods and principles of experimental sciences founded on computational methods. We study the use of machines to perform scientific tasks, with particular reference to computer models and simulations. Each part aims at defining a notion of computational validity according to the corresponding methodological approach--

a discipline of programming: Integrated Formal Methods Dominique Méry, Stephan Merz, 2010-10-01 Annotation. This book constitutes the refereed proceedings of the 8th International Conference on Integrated Formal Methods, IFM 2010, held in Nancy, France, in October 2010. The 20 revised full papers presented together with 3 invited papers were carefully reviewed and selected from 59 submissions. The papers address the spectrum of integrated formal methods, ranging from formal and semiformal notations, semantics, refinement, verification and model transformations to type systems, logics, tools and case studies.

a discipline of programming: Teaching and Learning Formal Methods C. Neville Dean, Michael G. Hinchey, 1996-09-17 As computer systems continue to advance, the positions they hold in human society continue to gain power. Computers now control the flight of aircraft, the cooling systems in chemical plants, and feedback loops in nuclear reactors. Because of the vital roles these systems play, there has been growing concern about the reliability and safety of these advanced computers. Formal methods are now widely recognized as the most successful means of assuring the reliability of complex computer systems. Because formal methods are being mandated in more and more international standards, it is critical that engineers, managers, and industrial project leaders are well trained and conversant in the application of these methods. This book covers a broad range of issues relating to the pedagogy of formal methods. The contributors, all acknowledged experts, have based their contributions on extensive experiences teaching and applying formal methods in both academia and industry.The two editors, both well known in this area, propose various techniques that can help to dismiss myths that formal methods are difficult to use and hard to learn. Teaching and Learning Formal Methods will be an indispensable text for educators in the fields of computer science, mathematics, software engineering, and electronic engineering as well as to management and product leaders concerned with trainingrecent graduates. Offers proven methods for teaching formal methods, even to students who lack a strong background in mathematics Addresses the important role that formal methods play in society and considers their growing future potential Includes contributions from several pioneers in the area Features a foreword written by Edsger W. Dijkstra

a discipline of programming: Perspectives on Software Documentation Thomas Barker, 2020-11-25 This book is designed to address the randomness of the literature on software documentation. As anyone interested in software documentation is aware, the field is highly synthetic; information about software documentation may be found in engineering, computer science training, technical communication, management, education and so on. Perspectives on Software Documentation contains a variety of perspectives, all tied together by the shared need to make software products more usable.

a discipline of programming: Theories of Programming and Formal Methods Jonathan P. Bowen, Qin Li, Qiwen Xu, 2023-09-07 This Festschrift volume, dedicated to Jifeng He on the occasion of his 80th birthday, includes refereed papers by leading researchers, many of them current and former colleagues, presented at a dedicated celebration in the Shanghai Science Hall in

September 2023. Jifeng was an important researcher on the European ESPRIT ProCoS project and the Working Group on Provably Correct Systems, subsequently he collaborated with Tony Hoare on Unifying Theories of Programming. Jifeng returned to China in 1998, first to the United Nations University in Macau and then to the East China Normal University in Shanghai. He has since founded an Artificial Intelligence research institute that focuses on the application of technology in large-scale industrial software systems. His scientific contributions have been recognized through his election to membership of the Chinese Academy of Sciences. The first paper in the volume provides an overview of Jifeng's research contributions, especially in the area of formal methods, and the following two papers detail developments in UTP and rCOS (refinement calculus of object systems). In the next two sections of the book, the editors included papers by colleagues and coauthors of Jifeng while he was at the University of Oxford and engaged with the European ProCoS project. The section that follows includes papers authored by colleagues from his later research in China and Europe. The final section includes a paper related to Jifeng's recent roadmap for UTP.

Related to a discipline of programming

DISCIPLINE Definition & Meaning - Merriam-Webster discipline implies training in habits of order and precision. school implies training or disciplining especially in what is hard to master

Discipline - Wikipedia Discipline entails executing habits precisely as intended, enhancing the likelihood of accomplishment and overcoming competing behaviors. Acting promptly exemplifies discipline,

DISCIPLINE | English meaning - Cambridge Dictionary DISCIPLINE definition: 1. training that makes people more willing to obey or more able to control themselves, often in the. Learn more

DISCIPLINE Definition & Meaning | Discipline definition: training to act in accordance with rules; drill.. See examples of DISCIPLINE used in a sentence

Discipline: Definition, Meaning, and Examples Discipline (verb): To train or control by enforcing obedience or self-control, often through corrective measures. The term "discipline" spans a variety of meanings, from the

discipline noun - Definition, pictures, pronunciation and Definition of discipline noun in Oxford Advanced Learner's Dictionary. Meaning, pronunciation, picture, example sentences, grammar, usage notes, synonyms and more

Why Discipline Matters—and 5 Ways to Work On It Discipline is essential in the change process, because you'll need to keep new behaviors in place even after you've met your initial goals. Many people never learned to be

DISCIPLINE Definition & Meaning - Merriam-Webster discipline implies training in habits of order and precision. school implies training or disciplining especially in what is hard to master

Discipline - Wikipedia Discipline entails executing habits precisely as intended, enhancing the likelihood of accomplishment and overcoming competing behaviors. Acting promptly exemplifies discipline,

DISCIPLINE | English meaning - Cambridge Dictionary DISCIPLINE definition: 1. training that makes people more willing to obey or more able to control themselves, often in the. Learn more

DISCIPLINE Definition & Meaning | Discipline definition: training to act in accordance with rules; drill.. See examples of DISCIPLINE used in a sentence

Discipline: Definition, Meaning, and Examples Discipline (verb): To train or control by enforcing obedience or self-control, often through corrective measures. The term "discipline" spans a variety of meanings, from the

discipline noun - Definition, pictures, pronunciation and Definition of discipline noun in Oxford Advanced Learner's Dictionary. Meaning, pronunciation, picture, example sentences, grammar, usage notes, synonyms and more

Why Discipline Matters—and 5 Ways to Work On It Discipline is essential in the change process, because you'll need to keep new behaviors in place even after you've met your initial goals. Many people never learned to be

DISCIPLINE Definition & Meaning - Merriam-Webster discipline implies training in habits of order and precision. school implies training or disciplining especially in what is hard to master

Discipline - Wikipedia Discipline entails executing habits precisely as intended, enhancing the likelihood of accomplishment and overcoming competing behaviors. Acting promptly exemplifies discipline,

DISCIPLINE | English meaning - Cambridge Dictionary DISCIPLINE definition: 1. training that makes people more willing to obey or more able to control themselves, often in the. Learn more

DISCIPLINE Definition & Meaning | Discipline definition: training to act in accordance with rules; drill.. See examples of DISCIPLINE used in a sentence

Discipline: Definition, Meaning, and Examples Discipline (verb): To train or control by enforcing obedience or self-control, often through corrective measures. The term "discipline" spans a variety of meanings, from the

discipline noun - Definition, pictures, pronunciation and Definition of discipline noun in Oxford Advanced Learner's Dictionary. Meaning, pronunciation, picture, example sentences, grammar, usage notes, synonyms and more

Why Discipline Matters—and 5 Ways to Work On It Discipline is essential in the change process, because you'll need to keep new behaviors in place even after you've met your initial goals. Many people never learned to be

DISCIPLINE Definition & Meaning - Merriam-Webster discipline implies training in habits of order and precision. school implies training or disciplining especially in what is hard to master

Discipline - Wikipedia Discipline entails executing habits precisely as intended, enhancing the likelihood of accomplishment and overcoming competing behaviors. Acting promptly exemplifies discipline,

DISCIPLINE | English meaning - Cambridge Dictionary DISCIPLINE definition: 1. training that makes people more willing to obey or more able to control themselves, often in the. Learn more

DISCIPLINE Definition & Meaning | Discipline definition: training to act in accordance with rules; drill.. See examples of DISCIPLINE used in a sentence

Discipline: Definition, Meaning, and Examples Discipline (verb): To train or control by enforcing obedience or self-control, often through corrective measures. The term "discipline" spans a variety of meanings, from the

discipline noun - Definition, pictures, pronunciation and Definition of discipline noun in Oxford Advanced Learner's Dictionary. Meaning, pronunciation, picture, example sentences, grammar, usage notes, synonyms and more

Why Discipline Matters—and 5 Ways to Work On It Discipline is essential in the change process, because you'll need to keep new behaviors in place even after you've met your initial goals. Many people never learned to be

DISCIPLINE Definition & Meaning - Merriam-Webster discipline implies training in habits of order and precision. school implies training or disciplining especially in what is hard to master

Discipline - Wikipedia Discipline entails executing habits precisely as intended, enhancing the likelihood of accomplishment and overcoming competing behaviors. Acting promptly exemplifies discipline,

DISCIPLINE | English meaning - Cambridge Dictionary DISCIPLINE definition: 1. training that makes people more willing to obey or more able to control themselves, often in the. Learn more

DISCIPLINE Definition & Meaning | Discipline definition: training to act in accordance with rules; drill.. See examples of DISCIPLINE used in a sentence

Discipline: Definition, Meaning, and Examples Discipline (verb): To train or control by enforcing obedience or self-control, often through corrective measures. The term "discipline" spans a variety of meanings, from the

discipline noun - Definition, pictures, pronunciation and Definition of discipline noun in Oxford Advanced Learner's Dictionary. Meaning, pronunciation, picture, example sentences, grammar, usage notes, synonyms and more

Why Discipline Matters—and 5 Ways to Work On It Discipline is essential in the change process, because you'll need to keep new behaviors in place even after you've met your initial goals. Many people never learned to be

Back to Home: http://www.speargroupllc.com