

## Building Evolutionary Architectures Support Constant Change

In *The Selfish Gene*, Richard Dawkins crystallized the gene's eye view of evolution developed by W.D. Hamilton and others. The book provoked widespread and heated debate. Written in part as a response, *The Extended Phenotype* gave a deeper clarification of the central concept of the gene as the unit of selection; but it did much more besides. In it, Dawkins extended the gene's eye view to argue that the genes that sit within an organism have an influence that reaches out beyond the visible traits in that body - the phenotype - to the wider environment, which can include other individuals. So, for instance, the genes of the beaver drive it to gather twigs to produce the substantial physical structure of a dam; and the genes of the cuckoo chick produce effects that manipulate the behaviour of the host bird, making it nurture the intruder as one of its own. This notion of the extended phenotype has proved to be highly influential in the way we understand evolution and the natural world. It represents a key scientific contribution to evolutionary biology, and it continues to play an important role in research in the life sciences. *The Extended Phenotype* is a conceptually deep book that forms important reading for biologists and students. But Dawkins' clear exposition is accessible to all who are prepared to put in a little effort. Oxford Landmark Science books are 'must-read' classics of modern science writing which have crystallized big ideas, and shaped the way we think.

Are you working on a codebase where cost overruns, death marches, and heroic fights with legacy code monsters are the norm? Battle these adversaries with novel ways to identify and prioritize technical debt, based on behavioral data from how developers work with code. And that's just for starters. Because good code involves social design, as well as technical design, you can find surprising dependencies between people and code to resolve coordination bottlenecks among teams. Best of all, the techniques build on behavioral data that you already have: your version-control system. Join the fight for better code! Use statistics and data science to uncover both problematic code and the behavioral patterns of the developers who build your software. This combination gives you insights you can't get from the code alone. Use these insights to prioritize refactoring needs, measure their effect, find implicit dependencies between different modules, and automatically create knowledge maps of your system based on actual code contributions. In a radical, much-needed change from common practice, guide organizational decisions with objective data by measuring how well your development teams align with the software architecture. Discover a comprehensive set of practical analysis techniques based on version-control data, where each point is illustrated with a case study from a real-world codebase. Because the techniques are language neutral, you can apply them to your own code no matter what programming language you use. Guide organizational decisions with objective data by measuring how well your development teams align with the software architecture. Apply research findings from social psychology to software development, ensuring you get the tools you need to coach your organization towards better code. If you're an experienced programmer, software architect, or technical manager, you'll get a new perspective that will change how you work with code. What You Need: You don't have to install anything to follow along in the book. The case studies in the book use well-known open source projects hosted on GitHub. You'll use CodeScene, a free software analysis tool for open source projects, for the case studies. We also discuss alternative tooling options where they exist.

Don't engineer by coincidence-design it like you mean it! Filled with practical techniques, *Design It!* is the perfect introduction to software architecture for programmers who are ready to grow their design skills. Lead your team as a software architect, ask the right stakeholders the right questions, explore design options, and help your team implement a system that promotes the right -ilities. Share your design decisions,

facilitate collaborative design workshops that are fast, effective, and fun-and develop more awesome software! With dozens of design methods, examples, and practical know-how, Design It! shows you how to become a software architect. Walk through the core concepts every architect must know, discover how to apply them, and learn a variety of skills that will make you a better programmer, leader, and designer. Uncover the big ideas behind software architecture and gain confidence working on projects big and small. Plan, design, implement, and evaluate software architectures and collaborate with your team, stakeholders, and other architects. Identify the right stakeholders and understand their needs, dig for architecturally significant requirements, write amazing quality attribute scenarios, and make confident decisions. Choose technologies based on their architectural impact, facilitate architecture-centric design workshops, and evaluate architectures using lightweight, effective methods. Write lean architecture descriptions people love to read. Run an architecture design studio, implement the architecture you've designed, and grow your team's architectural knowledge. Good design requires good communication. Talk about your software architecture with stakeholders using whiteboards, documents, and code, and apply architecture-focused design methods in your day-to-day practice. Hands-on exercises, real-world scenarios, and practical team-based decision-making tools will get everyone on board and give you the experience you need to become a confident software architect.

Microservices is an architectural style in which large, complex software applications are composed of one or more smaller services. Each of these microservices focuses on completing one task that represents a small business capability. These microservices can be developed in any programming language. They communicate with each other using language-neutral protocols, such as Representational State Transfer (REST), or messaging applications, such as IBM® MQ Light. This IBM Redbooks® publication gives a broad understanding of this increasingly popular architectural style, and provides some real-life examples of how you can develop applications using the microservices approach with IBM Bluemix™. The source code for all of these sample scenarios can be found on GitHub (<https://github.com/>). The book also presents some case studies from IBM products. We explain the architectural decisions made, our experiences, and lessons learned when redesigning these products using the microservices approach. Information technology (IT) professionals interested in learning about microservices and how to develop or redesign an application in Bluemix using microservices can benefit from this book.

A single dramatic software failure can cost a company millions of dollars - but can be avoided with simple changes to design and architecture. This new edition of the best-selling industry standard shows you how to create systems that run longer, with fewer failures, and recover better when bad things happen. New coverage includes DevOps, microservices, and cloud-native architecture. Stability antipatterns have grown to include systemic problems in large-scale systems. This is a must-have pragmatic guide to engineering for production systems. If you're a software developer, and you don't want to get alerts every night for the rest of your life, help is here. With a combination of case studies about huge losses - lost revenue, lost reputation, lost time, lost opportunity - and practical, down-to-earth advice that was all gained through painful experience, this book helps you avoid the pitfalls that cost companies millions of dollars in downtime and reputation. Eighty percent of project life-cycle cost is in production, yet few books address this topic. This updated edition deals with the production of today's systems - larger, more complex, and heavily virtualized - and includes information on chaos engineering, the discipline of applying randomness and deliberate stress to reveal systematic problems. Build systems that survive the real world, avoid downtime, implement zero-downtime upgrades and continuous delivery, and make cloud-native applications resilient. Examine ways to architect, design, and build software - particularly distributed systems - that stands up to the typhoon winds of a flash mob, a Slashdotting, or a link on Reddit. Take a hard look at software that failed the test and find ways to make sure your software survives. To skip the pain and get the experience...get this book.

## Acces PDF Building Evolutionary Architectures Support Constant Change

Describes ways to incorporate domain modeling into software development.

The Digital Age is having a broad and profound impact on companies and entire industries. Rather than simply automate or embed digital technology into existing offerings, your business needs to rethink everything. In this practical book, three ThoughtWorks professionals provide a game plan to help your business through this transformation, along with technical concepts that you need to know to be an effective leader in a modern digital business. Chock-full of practical advice and case studies that show how businesses have transitioned, this book reveals lessons learned in guiding companies through digital transformation. While there's no silver bullet available, you'll discover effective ways to create lasting change at your organization. With this book, you'll discover how to: Realign the business and operating architecture to focus on customer value Build a more responsive and agile organization to deal with speed and ambiguity Build next generation technology capability as a core differentiator

The practice of enterprise application development has benefited from the emergence of many new enabling technologies. Multi-tiered object-oriented platforms, such as Java and .NET, have become commonplace. These new tools and technologies are capable of building powerful applications, but they are not easily implemented. Common failures in enterprise applications often occur because their developers do not understand the architectural lessons that experienced object developers have learned. Patterns of Enterprise Application Architecture is written in direct response to the stiff challenges that face enterprise application developers. The author, noted object-oriented designer Martin Fowler, noticed that despite changes in technology--from Smalltalk to CORBA to Java to .NET--the same basic design ideas can be adapted and applied to solve common problems. With the help of an expert group of contributors, Martin distills over forty recurring solutions into patterns. The result is an indispensable handbook of solutions that are applicable to any enterprise application platform. This book is actually two books in one. The first section is a short tutorial on developing enterprise applications, which you can read from start to finish to understand the scope of the book's lessons. The next section, the bulk of the book, is a detailed reference to the patterns themselves. Each pattern provides usage and implementation information, as well as detailed code examples in Java or C#. The entire book is also richly illustrated with UML diagrams to further explain the concepts. Armed with this book, you will have the knowledge necessary to make important architectural decisions about building an enterprise application and the proven patterns for use when building them. The topics covered include · Dividing an enterprise application into layers · The major approaches to organizing business logic · An in-depth treatment of mapping between objects and relational databases · Using Model-View-Controller to organize a Web presentation · Handling concurrency for data that spans multiple transactions · Designing distributed object interfaces

In this truly unique technical book, today's leading software architects present valuable principles on key development issues that go way beyond technology. More than four dozen architects -- including Neal Ford, Michael Nygard, and Bill de hOra -- offer advice for communicating with stakeholders, eliminating complexity, empowering developers, and many more practical lessons they've learned from years of experience. Among the 97 principles in this book, you'll find useful advice such as: Don't Put Your Resume Ahead of the Requirements (Nitin Borwankar) Chances Are, Your Biggest Problem Isn't Technical (Mark Ramm) Communication Is King; Clarity and Leadership, Its Humble Servants (Mark Richards) Simplicity Before Generality, Use Before Reuse (Kevlin Henney) For the End User, the Interface Is the System (Vinayak Hegde) It's Never Too Early to Think About Performance (Rebecca Parsons) To be successful as a software architect, you need to master both business and technology. This book tells you what top software architects think is important and how they approach a project. If you want to enhance your career, 97 Things Every Software Architect Should Know is essential reading.

Many claims are made about how certain tools, technologies, and practices improve software development. But which claims are verifiable, and which are merely wishful thinking? In this book, leading thinkers such as Steve McConnell, Barry Boehm, and Barbara Kitchenham offer essays that uncover the truth and unmask myths commonly held among the software development community. Their insights may surprise you. Are some programmers really ten times more productive than others? Does writing tests first help you develop better code faster? Can code metrics predict the number of bugs in a piece of software? Do design patterns actually make better software? What effect does personality have on pair programming? What matters more: how far apart people are geographically, or how far apart they are in the org chart? Contributors include: Jorge Aranda Tom Ball Victor R. Basili Andrew Begel Christian Bird Barry Boehm Marcelo Cataldo Steven Clarke Jason Cohen Robert DeLine Madeline Diep Hakan Erdogmus Michael Godfrey Mark Guzdial Jo E. Hannay Ahmed E. Hassan Israel Herraiz Kim Sebastian Herzig Cory Kapser Barbara Kitchenham Andrew Ko Lucas Layman Steve McConnell Tim Menzies Gail Murphy Nachi Nagappan Thomas J. Ostrand Dewayne Perry Marian Petre Lutz Prechelt Rahul Premraj Forrest Shull Beth Simon Diomidis Spinellis Neil Thomas Walter Tichy Burak Turhan Elaine J. Weyuker Michele A. Whitecraft Laurie Williams Wendy M. Williams Andreas Zeller Thomas Zimmermann

Presentation Patterns is the first book on presentations that categorizes and organizes the building blocks (or patterns) that you'll need to communicate effectively using presentation tools like Keynote and PowerPoint. Patterns are like the lower-level steps found inside recipes; they are the techniques you must master to be considered a master chef or master presenter. You can use the patterns in this book to construct your own recipes for different contexts, such as business meetings, technical demonstrations, scientific expositions, and keynotes, just to name a few. Although there are no such things as antirecipes, this book shows you lots of antipatterns—things you should avoid doing in presentations. Modern presentation tools often encourage ineffective presentation techniques, but this book shows you how to avoid them. Each pattern is introduced with a memorable name, a definition, and a brief explanation of motivation. Readers learn where the pattern applies, the consequences of applying it, and how to apply it. The authors also identify critical antipatterns: clichés, fallacies, and design mistakes that cause presentations to disappoint. These problems are easy to avoid—once you know how. Presentation Patterns will help you Plan what you'll say, who you'll say it to, how long you'll talk, and where you'll present Perfectly calibrate your presentation to your audience Use the storyteller's "narrative arc" to full advantage Strengthen your credibility—and avoid mistakes that hurt it Hone your message before you ever touch presentation software Incorporate visuals that support your message instead of hindering it Create highly effective "infodecks" that work when you're not able to deliver a talk in person Construct slides that really communicate and avoid "Ant Fonts," "Floodmarks," "Alienating Artifacts," and other errors Master 13 powerful techniques for delivering your presentation with power, authority, and clarity Whether you use this book as a handy reference or read it from start to finish, it will be a revelation: an entirely new language for systematically planning, creating, and delivering more powerful presentations. You'll quickly find it indispensable—no matter what you're presenting, who your audiences are, or what message you're driving home.

Software architecture—the conceptual glue that holds every phase of a project together for its many stakeholders—is widely recognized as a critical element in modern software development. Practitioners have increasingly discovered that close attention to a software system's architecture pays valuable dividends. Without an architecture that is appropriate for the problem being solved, a project will stumble along or, most likely, fail. Even with a superb architecture, if that architecture is not well understood or well communicated the project is unlikely to succeed. Documenting Software Architectures, Second Edition, provides the most complete and current guidance, independent of language

or notation, on how to capture an architecture in a commonly understandable form. Drawing on their extensive experience, the authors first help you decide what information to document, and then, with guidelines and examples (in various notations, including UML), show you how to express an architecture so that others can successfully build, use, and maintain a system from it. The book features rules for sound documentation, the goals and strategies of documentation, architectural views and styles, documentation for software interfaces and software behavior, and templates for capturing and organizing information to generate a coherent package. New and improved in this second edition: Coverage of architectural styles such as service-oriented architectures, multi-tier architectures, and data models Guidance for documentation in an Agile development environment Deeper treatment of documentation of rationale, reflecting best industrial practices Improved templates, reflecting years of use and feedback, and more documentation layout options A new, comprehensive example (available online), featuring documentation of a Web-based service-oriented system Reference guides for three important architecture documentation languages: UML, AADL, and SySML

Annotation Over the past 10 years, distributed systems have become more fine-grained. From the large multi-million line long monolithic applications, we are now seeing the benefits of smaller self-contained services. Rather than heavy-weight, hard to change Service Oriented Architectures, we are now seeing systems consisting of collaborating microservices. Easier to change, deploy, and if required retire, organizations which are in the right position to take advantage of them are yielding significant benefits. This book takes an holistic view of the things you need to be cognizant of in order to pull this off. It covers just enough understanding of technology, architecture, operations and organization to show you how to move towards finer-grained systems.

A book for Tech Leads, from Tech Leads. Discover how more than 35 Tech Leads find the delicate balance between the technical and non-technical worlds. Discover the challenges a Tech Lead faces and how to overcome them. You may be surprised by the lessons they have to share.

Architects are often harried because they have no clean, easy decisions: everything is an awful tradeoff between two or more less than perfect alternatives. These are the difficult problems architects face, what this book's authors call "the hard parts." These topics have no best practices, forcing architects to understand various tradeoffs to succeed. This book discusses these hard parts by not only investigating what makes architecture so difficult, but also by providing proven ways to address these problems and make them easier. The book explores topics such as choosing an appropriate architecture, deciding on service granularity, managing workflows and orchestration, managing and decoupling contracts, managing distributed transactions, and optimizing operational characteristics such as scalability, elasticity, and performance. As practicing consultants, the authors focus on questions they commonly hear architects ask and provide techniques that enable them to discover the tradeoffs necessary to answer these questions.

Despite the buzz surrounding the cloud computing, only a small percentage of organizations have actually deployed this new style of IT—so far. If you're planning your long-term cloud strategy, this practical book provides insider knowledge and actionable real-world lessons regarding planning, design, operations, security, and application transformation. This book teaches business and technology managers how to transition their organization's traditional IT to cloud computing. Rather than yet another book trying to sell or convince readers on the benefits of clouds, this book provides guidance, lessons learned, and best practices on how to design, deploy, operate, and secure an enterprise cloud based on real-world experience. Author James Bond provides useful guidance and best-practice checklists based on his field experience with real customers and cloud providers. You'll view cloud services from the perspective of a consumer and as an

owner/operator of an enterprise private or hybrid cloud, and learn valuable lessons from successful and less-than-successful organization use-case scenarios. This is the information every CIO needs in order to make the business and technical decisions to finally execute on their journey to cloud computing. Get updated trends and definitions in cloud computing, deployment models, and for building or buying cloud services Discover challenges in cloud operations and management not foreseen by early adopters Use real-world lessons to plan and build an enterprise private or hybrid cloud Learn how to assess, port, and migrate legacy applications to the cloud Identify security threats and vulnerabilities unique to the cloud Employ a cloud management system for your enterprise (private or multi-provider hybrid) cloud ecosystem Understand the challenges for becoming an IT service broker leveraging the power of the cloud

Learn the principles of good software design, and how to turn those principles into great code. This book introduces you to software engineering — from the application of engineering principles to the development of software. You'll see how to run a software development project, examine the different phases of a project, and learn how to design and implement programs that solve specific problems. It's also about code construction — how to write great programs and make them work. Whether you're new to programming or have written hundreds of applications, in this book you'll re-examine what you already do, and you'll investigate ways to improve. Using the Java language, you'll look deeply into coding standards, debugging, unit testing, modularity, and other characteristics of good programs. With *Software Development, Design and Coding*, author and professor John Dooley distills his years of teaching and development experience to demonstrate practical techniques for great coding. What You'll Learn Review modern agile methodologies including Scrum and Lean programming Leverage the capabilities of modern computer systems with parallel programming Work with design patterns to exploit application development best practices Use modern tools for development, collaboration, and source code controls Who This Book Is For Early career software developers, or upper-level students in software engineering courses

Discover BIM: A better way to build better buildings Building Information Modeling (BIM) offers a novel approach to design, construction, and facility management in which a digital representation of the building product and process is used to facilitate the exchange and interoperability of information in digital format. BIM is beginning to change the way buildings look, the way they function, and the ways in which they are designed and built. The *BIM Handbook, Third Edition* provides an in-depth understanding of BIM technologies, the business and organizational issues associated with its implementation, and the profound advantages that effective use of BIM can provide to all members of a project team. Updates to this edition include: Information on the ways in which professionals should use BIM to gain maximum value New topics such as collaborative working, national and major construction clients, BIM standards and guides A discussion on how various professional roles have expanded through the widespread use and the new avenues of BIM practices and services A wealth of new case studies that clearly illustrate exactly how BIM is applied in a wide variety of conditions Painting a colorful and thorough picture of the state of the art in building information modeling, the *BIM Handbook, Third Edition* guides readers to successful implementations, helping them to avoid needless frustration and costs and take full advantage of this paradigm-shifting approach to construct better buildings that consume fewer materials and require less

time, labor, and capital resources.

Technologists who want their ideas heard, understood, and funded are often told to speak the language of business—without really knowing what that is. This book's toolkit provides architects, product managers, technology managers, and executives with a shared language—in the form of repeatable, practical patterns and templates—to produce great technology strategies. Author Eben Hewitt developed 39 patterns over the course of a decade in his work as CTO, CIO, and chief architect for several global tech companies. With these proven tools, you can define, create, elaborate, refine, and communicate your architecture goals, plans, and approach in a way that executives can readily understand, approve, and execute. This book covers: Architecture and strategy: Adopt a strategic architectural mindset to make a meaningful material impact Creating your strategy: Define the components of your technology strategy using proven patterns Communicating the strategy: Convey your technology strategy in a compelling way to a variety of audiences Bringing it all together: Employ patterns individually or in clusters for specific problems; use the complete framework for a comprehensive strategy

Evolutionary architecture attempts to evolve form and structure in emulation of the evolutionary processes of nature. It considers architecture as a form of artificial life. This approach has formed the basis for the author's teaching programme for AA Diploma Unit II.

This book uncovers the guiding principles behind Tsui's evolutionary approach to explore the many design lessons that can be learned from nature and share the impressive results of their application to architectural projects.

This is a practical guide for software developers, and different than other software architecture books. Here's why: It teaches risk-driven architecting. There is no need for meticulous designs when risks are small, nor any excuse for sloppy designs when risks threaten your success. This book describes a way to do just enough architecture. It avoids the one-size-fits-all process tar pit with advice on how to tune your design effort based on the risks you face. It democratizes architecture. This book seeks to make architecture relevant to all software developers. Developers need to understand how to use constraints as guiderails that ensure desired outcomes, and how seemingly small changes can affect a system's properties. It cultivates declarative knowledge. There is a difference between being able to hit a ball and knowing why you are able to hit it, what psychologists refer to as procedural knowledge versus declarative knowledge. This book will make you more aware of what you have been doing and provide names for the concepts. It emphasizes the engineering. This book focuses on the technical parts of software development and what developers do to ensure the system works not job titles or processes. It shows you how to build models and analyze architectures so that you can make principled design tradeoffs. It describes the techniques software designers use to reason about medium to large sized problems and points out where you can learn specialized techniques in more detail. It provides practical advice. Software design decisions influence the architecture and vice versa. The approach in this book embraces drill-down/pop-up behavior by describing models that have various levels of abstraction, from architecture to data structure design.

Authored by two of the leading authorities in the field, this guide offers readers the knowledge and skills needed to achieve

proficiency with embedded software.

An ethologist shows man to be a gene machine whose world is one of savage competition and deceit

Salary surveys worldwide regularly place software architect in the top 10 best jobs, yet no real guide exists to help developers become architects. Until now. This book provides the first comprehensive overview of software architecture's many aspects. Aspiring and existing architects alike will examine architectural characteristics, architectural patterns, component determination, diagramming and presenting architecture, evolutionary architecture, and many other topics. Mark Richards and Neal Ford—hands-on practitioners who have taught software architecture classes professionally for years—focus on architecture principles that apply across all technology stacks. You'll explore software architecture in a modern light, taking into account all the innovations of the past decade. This book examines: Architecture patterns: The technical basis for many architectural decisions Components: Identification, coupling, cohesion, partitioning, and granularity Soft skills: Effective team management, meetings, negotiation, presentations, and more Modernity: Engineering practices and operational approaches that have changed radically in the past few years Architecture as an engineering discipline: Repeatable results, metrics, and concrete valuations that add rigor to software architecture

As the digital economy changes the rules of the game for enterprises, the role of software and IT architects is also transforming. Rather than focus on technical decisions alone, architects and senior technologists need to combine organizational and technical knowledge to effect change in their company's structure and processes. To accomplish that, they need to connect the IT engine room to the penthouse, where the business strategy is defined. In this guide, author Gregor Hohpe shares real-world advice and hard-learned lessons from actual IT transformations. His anecdotes help architects, senior developers, and other IT professionals prepare for a more complex but rewarding role in the enterprise. This book is ideal for: Software architects and senior developers looking to shape the company's technology direction or assist in an organizational transformation Enterprise architects and senior technologists searching for practical advice on how to navigate technical and organizational topics CTOs and senior technical architects who are devising an IT strategy that impacts the way the organization works IT managers who want to learn what's worked and what hasn't in large-scale transformation

The software development ecosystem is constantly changing, providing a constant stream of new tools, frameworks, techniques, and paradigms. Over the past few years, incremental developments in core engineering practices for software development have created the foundations for rethinking how architecture changes over time, along with ways to protect important architectural characteristics as it evolves. This practical guide ties those parts together with a new way to think about architecture and time.

Gain insight into how hexagonal architecture can help to keep the cost of development low over the complete lifetime of an application Key Features Explore ways to make your software flexible, extensible, and adaptable Learn new concepts that you can easily blend with your own software development style Develop the mindset of building maintainable solutions instead of taking shortcuts Book Description We would all like to build software architecture that yields adaptable and flexible software with low development costs. But, unreasonable deadlines and shortcuts make it very hard to create such an architecture. Get Your Hands Dirty on Clean Architecture starts with a discussion about the conventional layered architecture style and its disadvantages. It also talks about the advantages of the domain-centric architecture styles of Robert C. Martin's Clean Architecture and Alistair Cockburn's Hexagonal Architecture. Then, the book dives into hands-on chapters that show you how to manifest a hexagonal architecture in actual code. You'll learn in detail about different mapping strategies between the layers of a



hexagonal architecture and see how to assemble the architecture elements into an application. The later chapters demonstrate how to enforce architecture boundaries. You'll also learn what shortcuts produce what types of technical debt and how, sometimes, it is a good idea to willingly take on those debts. After reading this book, you'll have all the knowledge you need to create applications using the hexagonal architecture style of web development. What you will learn

- Identify potential shortcomings of using a layered architecture
- Apply methods to enforce architecture boundaries
- Find out how potential shortcuts can affect the software architecture
- Produce arguments for when to use which style of architecture
- Structure your code according to the architecture
- Apply various types of tests that will cover each element of the architecture

Who this book is for This book is for you if you care about the architecture of the software you are building. To get the most out of this book, you must have some experience with web development. The code examples in this book are in Java. If you are not a Java programmer but can read object-oriented code in other languages, you will be fine. In the few places where Java or framework specifics are needed, they are thoroughly explained.

Building Evolutionary Architectures Support Constant Change"O'Reilly Media, Inc."

Through a critical study of issues such as order, form, space, style, place-making, aesthetics, and architectural theory, students are encouraged to think about their own creative ideas. The use of analytical reasoning, lateral thinking, drawing and modelling is emphasised. One of the biggest challenges for organizations that have adopted microservice architecture is the lack of architectural, operational, and organizational standardization. After splitting a monolithic application or building a microservice ecosystem from scratch, many engineers are left wondering what's next. In this practical book, author Susan Fowler presents a set of microservice standards in depth, drawing from her experience standardizing over a thousand microservices at Uber. You'll learn how to design microservices that are stable, reliable, scalable, fault tolerant, performant, monitored, documented, and prepared for any catastrophe. Explore production-readiness standards, including:

- Stability and Reliability: develop, deploy, introduce, and deprecate microservices; protect against dependency failures
- Scalability and Performance: learn essential components for achieving greater microservice efficiency
- Fault Tolerance and Catastrophe Preparedness: ensure availability by actively pushing microservices to fail in real time
- Monitoring: learn how to monitor, log, and display key metrics; establish alerting and on-call procedures
- Documentation and Understanding: mitigate tradeoffs that come with microservice adoption, including organizational sprawl and technical debt

In the race to compete in today's fast-moving markets, large enterprises are busy adopting new technologies for creating new products, processes, and business models. But one obstacle on the road to digital transformation is placing too much emphasis on technology, and not enough on the types of processes technology enables. What if different lines of business could build their own services and applications—and decision-making was distributed rather than centralized? This report explores the concept of a digital business platform as a way of empowering individual business sectors to act on data in real time. Much innovation in a digital enterprise will increasingly happen at the edge, whether it involves business users (from marketers to data scientists) or IoT devices. To facilitate the process, your core IT team can provide these sectors with the digital tools they need to innovate quickly. This report explores:

- Key cultural and organizational changes for developing business capabilities through cross-functional product teams
- A platform for integrating applications, data sources, business partners, clients, mobile apps, social networks, and IoT devices
- Creating internal API programs for building innovative edge services in low-code or no-code environments
- Tools including Integration Platform as a Service, Application Platform as a Service, and Integration Software as a Service
- The challenge of integrating microservices and serverless architectures
- Event-driven architectures for processing and reacting

to events in real time You'll also learn about a complete pervasive integration solution as a core component of a digital business platform to serve every audience in your organization.

Refactoring has proven its value in a wide range of development projects—helping software professionals improve system designs, maintainability, extensibility, and performance. Now, for the first time, leading agile methodologist Scott Ambler and renowned consultant Pramodkumar Sadalage introduce powerful refactoring techniques specifically designed for database systems. Ambler and Sadalage demonstrate how small changes to table structures, data, stored procedures, and triggers can significantly enhance virtually any database design—without changing semantics. You'll learn how to evolve database schemas in step with source code—and become far more effective in projects relying on iterative, agile methodologies. This comprehensive guide and reference helps you overcome the practical obstacles to refactoring real-world databases by covering every fundamental concept underlying database refactoring. Using start-to-finish examples, the authors walk you through refactoring simple standalone database applications as well as sophisticated multi-application scenarios. You'll master every task involved in refactoring database schemas, and discover best practices for deploying refactorings in even the most complex production environments. The second half of this book systematically covers five major categories of database refactorings. You'll learn how to use refactoring to enhance database structure, data quality, and referential integrity; and how to refactor both architectures and methods. This book provides an extensive set of examples built with Oracle and Java and easily adaptable for other languages, such as C#, C++, or VB.NET, and other databases, such as DB2, SQL Server, MySQL, and Sybase. Using this book's techniques and examples, you can reduce waste, rework, risk, and cost—and build database systems capable of evolving smoothly, far into the future.

If you're familiar with functional programming basics and want to gain a much deeper understanding, this in-depth guide takes you beyond syntax and demonstrates how you need to think in a new way. Software architect Neal Ford shows intermediate to advanced developers how functional coding allows you to step back a level of abstraction so you can see your programming problem with greater clarity. Each chapter shows you various examples of functional thinking, using numerous code examples from Java 8 and other JVM languages that include functional capabilities. This book may bend your mind, but you'll come away with a much better grasp of functional programming concepts. Understand why many imperative languages are adding functional capabilities Compare functional and imperative solutions to common problems Examine ways to cede control of routine chores to the runtime Learn how memoization and laziness eliminate hand-crafted solutions Explore functional approaches to design patterns and code reuse View real-world examples of functional thinking with Java 8, and in functional architectures and web frameworks Learn the pros and cons of living in a paradigmatically richer world If you're new to functional programming, check out Josh Backfield's book *Becoming Functional*.

Every day, companies struggle to scale critical applications. As traffic volume and data demands increase, these applications become more complicated and brittle, exposing risks and compromising availability. With the popularity of software as a service, scaling has never been more important. Updated with an expanded focus on modern architecture paradigms such as microservices and cloud computing, this practical guide provides techniques for building systems that can handle huge quantities of traffic, data, and demand—without affecting the quality your customers expect. Architects, managers, and directors in engineering and operations organizations will learn how to build applications at scale that run more smoothly and reliably to meet the needs of customers. Learn how scaling affects the availability of your services, why that matters, and how to improve it Dive into a modern service-based application architecture that ensures high availability and reduces the effects of service failures Explore the Single Team Owned Service Architecture paradigm (STOSA)—a model for scaling your

## Acces PDF Building Evolutionary Architectures Support Constant Change

development organization in tandem with your application Understand, measure, and mitigate risk in your systems Use the cloud to build highly scalable applications

Anyone who develops software for a living needs a proven way to produce it better, faster, and cheaper. The Productive Programmer offers critical timesaving and productivity tools that you can adopt right away, no matter what platform you use. Master developer Neal Ford not only offers advice on the mechanics of productivity-how to work smarter, spurn interruptions, get the most out your computer, and avoid repetition-he also details valuable practices that will help you elude common traps, improve your code, and become more valuable to your team. You'll learn to: Write the test before you write the code Manage the lifecycle of your objects fastidiously Build only what you need now, not what you might need later Apply ancient philosophies to software development Question authority, rather than blindly adhere to standards Make hard things easier and impossible things possible through meta-programming Be sure all code within a method is at the same level of abstraction Pick the right editor and assemble the best tools for the job This isn't theory, but the fruits of Ford's real-world experience as an Application Architect at the global IT consultancy ThoughtWorks. Whether you're a beginner or a pro with years of experience, you'll improve your work and your career with the simple and straightforward principles in The Productive Programmer.

An overview of the field of information organization that examines resource description as both a product and process of the contemporary digital environment. This timely book employs the unifying mechanism of the semantic web and the resource description framework to integrate the various traditions and practices of information and knowledge organization. Uniquely, it covers both the domain-specific traditions and practices and the practices of the 'metadata movement' through a single lens – that of resource description in the broadest, semantic web sense. This approach more readily accommodates coverage of the new Resource Description and Access (RDA) standard, which aims to move library cataloguing into the centre of the semantic web. The work surrounding RDA looks set to revolutionise the field of information organization, and this book will bring both the standard and its model and concepts into focus. Key topics include: • information resource attributes • metadata for information retrieval • metadata sources and quality • economics and management of metadata • knowledge organization systems • the semantic web • books and e-books, websites and audiovisual resources • business and government documents • learning resources • the field of information/knowledge organization. Readership: LIS students taking information organization courses at undergraduate and postgraduate levels, information professionals wishing to specialise in the metadata area, and existing metadata specialists who wish to update their knowledge.

The overwhelming majority of a software system's lifespan is spent in use, not in design or implementation. So, why does conventional wisdom insist that software engineers focus primarily on the design and development of large-scale computing systems? In this collection of essays and articles, key members of Google's Site Reliability Team explain how and why their commitment to the entire lifecycle has enabled the company to successfully build, deploy, monitor, and maintain some of the largest software systems in the world. You'll learn the principles and practices that enable Google engineers to make systems more scalable, reliable, and efficient—lessons directly applicable to your organization. This book is divided into four sections: Introduction—Learn what site reliability engineering is and why it differs from conventional IT industry practices Principles—Examine the patterns, behaviors, and areas of concern that influence the work of a site reliability engineer (SRE) Practices—Understand the theory and practice of an SRE's day-to-day work: building and operating large distributed computing systems Management—Explore Google's best practices for training, communication, and meetings that your organization can use How do you detangle a monolithic system and migrate it to a microservice architecture? How do you do it while maintaining business-as-

usual? As a companion to Sam Newman's extremely popular Building Microservices, this new book details a proven method for transitioning an existing monolithic system to a microservice architecture. With many illustrative examples, insightful migration patterns, and a bevy of practical advice to transition your monolith enterprise into a microservice operation, this practical guide covers multiple scenarios and strategies for a successful migration, from initial planning all the way through application and database decomposition. You'll learn several tried and tested patterns and techniques that you can use as you migrate your existing architecture. Ideal for organizations looking to transition to microservices, rather than rebuild Helps companies determine whether to migrate, when to migrate, and where to begin Addresses communication, integration, and the migration of legacy systems Discusses multiple migration patterns and where they apply Provides database migration examples, along with synchronization strategies Explores application decomposition, including several architectural refactoring patterns Delves into details of database decomposition, including the impact of breaking referential and transactional integrity, new failure modes, and more

[Copyright: d13777de1c75f0d8a70f744335fd8b9a](#)