# **Samstraumr: A Systems-Oriented Framework for Navigating Software Complexity**

**I. Introduction: Unveiling Samstraumr - A Multi-faceted Framework**

The landscape of modern software development is increasingly characterized by intricate systems demanding sophisticated design approaches. Traditional methodologies often struggle to keep pace with this growing complexity, sometimes leading to solutions that are difficult to maintain and evolve.1 The accumulation of technical debt, where expediency is prioritized over code quality, results in future rework and escalating maintenance costs.2 Samstraumr emerges as a design framework centered around Tube-Based Development (TBD), offering a structured approach to address these challenges.1 This report aims to provide a comprehensive analysis of Samstraumr's design model, its inherent features, the anticipated outcomes of its application, and the potential issues it seeks to prevent. This analysis will be conducted through philosophical, conceptual, and practical lenses to cater to a diverse audience encompassing academics, technology professionals, QA teams, scientists, and lawyers. Key themes explored within this report include the notion that adopting Samstraumr, while potentially a more demanding initial undertaking, ultimately leads to more comprehensive and valuable results. Furthermore, the report will delve into the analogy of "using the eternal now as a surfboard on a wave" to understand Samstraumr's dynamic nature and examine its relationship with established programming paradigms such as Object-Oriented Programming (OOP) and Functional Programming (FP).

This report is structured to provide a multi-faceted understanding of Samstraumr for a broad spectrum of professionals. For academics in software design and related disciplines, it offers a novel framework for analysis and potential research. Technology professionals, including software developers and architects, will find insights into how Samstraumr's Tube-Based Development can be practically applied to build and manage complex systems. Quality Assurance teams will gain an understanding of how Samstraumr's design principles might impact testing strategies and code reliability. Scientists, particularly those involved in modeling intricate natural or artificial systems, will discover potential parallels and transferable concepts. Finally, lawyers may find relevance in the contractual and liability implications associated with software developed using this framework. The report will first lay the architectural foundation of Tube-Based Development, followed by a philosophical interpretation of its core tenets. Subsequently, it will conceptualize Samstraumr within the broader context of software paradigms and systems thinking, culminating in an exploration of its practical resonance across various professional domains.

**II. The Architectural Foundation: Deconstructing Samstraumr's Tube-Based Development**

At the heart of Samstraumr lies the methodology of Tube-Based Development (TBD). This approach is predicated on the concept of interconnected tubes that process inputs and generate outputs.1 This design bears a resemblance to the idea of a pipeline in software engineering, where data undergoes a series of transformations as it flows through distinct processing stages.5 Software pipelines are characterized by their modularity, with each element performing a specific function, and this separation of concerns often contributes to improved maintainability.5 If TBD operates on similar principles, it likely inherits these advantages, fostering a more organized and comprehensible codebase. The documentation for Samstraumr, including Proposals Documentation and a Testing Strategy, is mentioned within the framework's repository.1 *However, access to these resources was not possible at the time of this analysis 1, which limits a more granular examination of the specific implementation details and testing protocols.*

The choice of the term "tube" as a central metaphor in this development framework is potentially significant. It might draw inspiration from the historical context of electron tubes (also known as vacuum tubes) in early electronics.7 Vacuum tubes played a crucial role in controlling the flow of electrons, acting as fundamental building blocks for amplification and switching.8 This historical analogy could suggest that Samstraumr's "tubes" are intended to provide a controlled and directed flow of data or information within the software system. Furthermore, the term "tube software" is used in industrial applications to describe software that manages processes and workflows related to physical tubes or pipelines, often in contexts like manufacturing and engineering.13 *Without access to the Samstraumr documentation, the precise nature of this connection remains speculative.*

Based on the limited information available, it is possible to infer some potential design principles and characteristics of Tube-Based Development. The emphasis on "interconnected tubes" suggests a modular architecture where different components of the system are linked together to achieve a larger goal. The notion of processing "inputs and outputs" implies clearly defined interfaces for each tube, facilitating the exchange of data between them. This modularity and the presence of well-defined interfaces are likely to contribute to the framework's stated aim of promoting maintainable code 1 by encouraging developers to separate concerns and create independent, reusable components. The mention of a dedicated "Testing Strategy" within the Samstraumr repository 1 further suggests that testability is a key consideration in the design of TBD, as a modular approach often allows for more straightforward unit testing of individual components.

When considering Samstraumr's relationship to existing software development models, its emphasis on building maintainable code faster 1 might align it more closely with iterative and agile methodologies, such as the Successive Approximation Model (SAM).17 SAM, in contrast to traditional linear approaches like ADDIE 18, emphasizes rapid feedback and continuous improvement through iterative design and development cycles.17 The focus on progress over perfection and incorporating feedback early and often resonates with the potential benefits suggested by Samstraumr's goals. While Trunk-Based Development (TBD) 20 also prioritizes rapid integration and a streamlined codebase, Samstraumr's use of the "tube" metaphor might indicate a different approach to modularity and organization compared to the feature branch-centric model of Trunk-Based Development.

**III. Navigating the Currents of Time: A Philosophical Interpretation of Samstraumr**

The decision to adopt a potentially more challenging design framework like Samstraumr carries philosophical implications related to the value of perseverance and long-term vision. Choosing a difficult path in software development often necessitates a greater initial investment of effort and learning.21 However, this upfront commitment can lead to a "richer end," characterized by enhanced code quality, improved maintainability, and greater overall system stability, potentially reducing the total cost of ownership over the software's lifecycle.21 This trade-off between immediate ease and enduring value mirrors the pursuit of "elegance" in software design.24 Crafting elegant code, defined by its clarity, simplicity, maintainability, and efficiency, often requires more thoughtful consideration and potentially more intricate initial implementations, yet it yields a more valuable and sustainable solution in the long run.24 This philosophy aligns with the brand ethos of companies such as Rolls Royce 28, which prioritize excellence, meticulous craftsmanship, and enduring quality over immediate convenience or ease of production, suggesting that a commitment to a more demanding approach can result in a superior final product.

The analogy of "using the eternal now as a surfboard on a wave" offers a profound philosophical lens through which to understand Samstraumr's approach to software development. The philosophical concept of the "eternal now" 32 emphasizes the significance of the present moment, advocating for a state of complete presence and engagement with the immediate reality. In the context of software, the "wave" can be interpreted as the continuous and often unpredictable flow of data, events, or changes in the system's state. "Surfing" on this wave then represents the ability of a system designed with Samstraumr to skillfully navigate this dynamic flow, adapting and responding to changes in real-time. This analogy strongly suggests a connection to the principles of reactive programming 36 and event-driven architectures.39 These paradigms are specifically designed to handle asynchronous data streams and facilitate real-time responses to events, allowing applications to react swiftly and efficiently to changes as they occur.38 The "eternal now" implies a constant state of readiness and responsiveness, mirroring the core tenets of reactive systems that aim to provide timely value to users by being resilient under failure and elastic under load.37

**IV. Conceptualizing Samstraumr: Bridging Software Paradigms through Systems Thinking**

Samstraumr's Tube-Based Development can be conceptualized within the context of established software development paradigms, particularly Object-Oriented Programming (OOP) and Functional Programming (FP), while also exhibiting a strong foundation in systems thinking. Object-Oriented Programming, with its emphasis on organizing software around "objects" that encapsulate data and behavior, shares some potential connections with TBD.43 Principles like encapsulation and modularity, central to OOP, might find parallels in the way individual "tubes" in Samstraumr are designed to be self-contained units with defined interfaces. However, the primary focus in TBD appears to be on the flow and processing of data through interconnected stages, rather than the state and behavior of discrete objects, suggesting a different approach to system decomposition compared to traditional OOP. While OOP offers flexibility through mechanisms like inheritance and polymorphism, it can sometimes lead to challenges such as complex inheritance hierarchies and tight coupling between objects.43 Samstraumr's modular tube-based architecture might offer an alternative way to manage complexity and promote looser coupling by emphasizing clear data flow and processing steps.

Functional Programming, which emphasizes functions as first-class citizens and promotes concepts like immutability and pure functions, also shows potential relationships with Samstraumr.47 The idea of "tubes" taking inputs and producing outputs aligns closely with the concept of functions in FP. If these tubes are designed to be stateless and operate on immutable data, where once a value is set, it remains constant 47, Samstraumr could strongly embody functional programming principles. This approach can lead to more predictable code, as functions produce the same output for the same input without side effects.47 The "pipeline" analogy used to describe TBD 5 also resonates with the concept of function composition in FP, where the output of one function becomes the input of the next, creating a chain of transformations. By adopting FP principles, Samstraumr has the potential to enhance testability and maintainability, as pure functions are easier to test in isolation and code with immutable data is less prone to unexpected state changes.47

Underlying Samstraumr's architectural approach is a clear application of systems theory.51 Systems theory emphasizes understanding complex entities as interconnected wholes, where the relationships and interactions between components are as important as the individual parts themselves.51 Samstraumr's core concept of interconnected tubes working collaboratively to process data directly reflects this holistic perspective. Key systems concepts such as interconnectedness 55 and interdependence 55 are fundamental to the TBD methodology, where the functionality of the overall system emerges from the interactions between individual tubes. The concept of boundaries 52 is also relevant, as each tube likely has defined inputs and outputs, establishing a clear interface with its environment within the system. Furthermore, understanding potential feedback mechanisms 52 within a Samstraumr system could be crucial for analyzing its dynamic behavior and ability to adapt to changing conditions. The framework might also exhibit emergent properties 52, where the integrated behavior of the interconnected tubes results in system-level characteristics that are not immediately obvious from examining the individual tubes in isolation. The historical evolution of systems theory across various disciplines, including biology 74, engineering 79, and social sciences 53, demonstrates its broad applicability to understanding organized complexity. Pioneers like Ludwig von Bertalanffy 55 and Norbert Wiener 92 laid the groundwork for this interdisciplinary field, emphasizing the importance of viewing systems as wholes. Alexander Bogdanov's "Tektology" 58, an early attempt at a universal science of organization, further underscores the long-standing interest in developing frameworks for understanding complex systems across diverse domains.

**V. Practical Resonance: Applications of Samstraumr Across Professional Domains**

Samstraumr's Tube-Based Development holds practical relevance for a wide range of professional domains, offering potential benefits and new perspectives. For academics, Samstraumr presents a compelling subject of study within the fields of software architecture, systems design, and programming paradigms. Its unique approach to software construction, blending concepts from different paradigms and grounded in systems thinking, makes it a valuable case study for exploring the integration of philosophical ideas with practical software engineering principles.

Technology professionals can leverage TBD to design and build software systems that are inherently modular, maintainable, and scalable. The clear separation of concerns promoted by the "tube" concept can lead to improved code quality and potentially faster development cycles by allowing teams to work on individual tubes independently. The well-defined interfaces between tubes can also enhance team collaboration and reduce the complexity of integrating different parts of the system. Samstraumr's TBD could be particularly applicable in projects involving complex data processing, distributed systems, and real-time applications where managing the flow of information is critical.

QA teams stand to benefit from the modularity of TBD, as it allows for more focused and isolated testing of individual tubes. This can simplify the testing process, improve test coverage, and potentially lead to a reduction in the number of bugs that make their way into the final product. The "Testing Strategy" documentation within the Samstraumr repository 1 is likely to provide specific guidelines and best practices for ensuring the quality of software developed using this framework.

Scientists involved in modeling complex systems across various disciplines, such as biology and ecology 101, might find conceptual parallels with Samstraumr's TBD. The framework's emphasis on interconnected components with defined interactions mirrors the way scientists often model natural systems, where understanding the relationships between different parts is essential for comprehending the overall system behavior. The "eternal now" analogy could also resonate with scientists studying dynamic systems and analyzing real-time data, where the focus is on understanding and responding to continuous streams of information.

For lawyers, the adoption of a framework like Samstraumr in software development projects raises several considerations. In drafting software development contracts, it would be important to clearly define the specifications and deliverables in terms of the "tubes" and their interconnections. The framework's emphasis on modularity and clear interfaces might also affect the determination of liability and responsibility in the event of system failures. Analogies could potentially be drawn to legal frameworks that rely on interconnected components and defined processes to achieve specific outcomes.

**VI. Conclusion: Embracing the Richness of a Challenging Path**

In conclusion, Samstraumr presents a unique and potentially valuable approach to software design through its Tube-Based Development methodology. While the initial learning curve and the shift in thinking required to fully embrace this framework might represent a more challenging path, the potential rewards in terms of code maintainability, scalability, and overall system robustness appear significant. The philosophical underpinnings of Samstraumr, emphasizing long-term value and responsiveness to dynamic data flows, align with principles of elegance in software design and the core tenets of reactive programming and event-driven architectures.

The conceptualization of Samstraumr within the broader landscape of software development reveals its capacity to bridge established paradigms like OOP and FP while firmly grounding itself in the principles of systems thinking. By viewing software as an interconnected system of processing units, Samstraumr encourages a holistic approach that considers the interactions and interdependencies between components. This perspective offers a powerful lens for tackling the increasing complexity of modern software. The practical resonance of Samstraumr across diverse professional domains, from academia and technology to quality assurance, science, and even law, underscores its potential to offer new insights and solutions in an increasingly interconnected and data-driven world. Embracing the richness of this challenging path could lead to the development of more resilient, adaptable, and ultimately more valuable software systems.

#### Works cited

1. Samstraumr/README.md at main · heymumford/Samstraumr · GitHub, accessed on April 3, 2025, <https://github.com/heymumford/Samstraumr/blob/main/README.md>
2. What is Tech Debt? Signs & How to Effectively Manage It | Atlassian, accessed on April 3, 2025, <https://www.atlassian.com/agile/software-development/technical-debt>
3. What is Technical Debt? Causes, Types & Definition Guide - Sonar, accessed on April 3, 2025, <https://www.sonarsource.com/learn/technical-debt/>
4. What Is Technical Debt? | Definition and Examples - ProductPlan, accessed on April 3, 2025, <https://www.productplan.com/glossary/technical-debt/>
5. Pipeline (software) - Wikipedia, accessed on April 3, 2025, <https://en.wikipedia.org/wiki/Pipeline_(software)>
6. heymumford/Samstraumr: Samstraumr helps you build ... - GitHub, accessed on April 3, 2025, <https://github.com/heymumford/Samstraumr>
7. Electron tube | Vacuum Tubes, Cathode Rays, & Applications | Britannica, accessed on April 3, 2025, <https://www.britannica.com/technology/electron-tube>
8. Vacuum tube - Wikipedia, accessed on April 3, 2025, <https://en.wikipedia.org/wiki/Vacuum_tube>
9. Vacuum Tubes: The World Before Transistors - Engineering.com, accessed on April 3, 2025, <https://www.engineering.com/vacuum-tubes-the-world-before-transistors/>
10. ᐉ What is The Vacuum Tube: Main Types, Pros and Cons, accessed on April 3, 2025, <https://vacuum-tubes.com/what-is-the-vacuum-tube/>
11. Exploration of Diode Valves and Vacuum Tubes: Operating Principles, Theoretical Insights, and Formulas - Allelco, accessed on April 3, 2025, <https://www.allelcoelec.com/blog/exploration-of-diode-valves-and-vacuum-tubes-operating-principles,theoretical-insights,and-formulas.html>
12. What is a Vacuum tube? | Lenovo US, accessed on April 3, 2025, <https://www.lenovo.com/us/en/glossary/vacuum-tube/>
13. Tube software - All industrial manufacturers - DirectIndustry, accessed on April 3, 2025, <https://www.directindustry.com/industrial-manufacturer/tube-software-110876.html>
14. Programming Tube | TRUMPF, accessed on April 3, 2025, <https://www.trumpf.com/en_US/products/software/programming-software/programming-tube/>
15. Development of Educational Software for Designing Shell-and-Tube Heat Exchangers | Request PDF - ResearchGate, accessed on April 3, 2025, <https://www.researchgate.net/publication/275685566_Development_of_Educational_Software_for_Designing_Shell-and-Tube_Heat_Exchangers>
16. Software Development for Thermal Design of Shell and Tube Heat Exchanger, accessed on April 3, 2025, <https://www.semanticscholar.org/paper/Software-Development-for-Thermal-Design-of-Shell-Upadhyay/d1a0be81cd5425ce9fb4367b52f8b0f142058ca3>
17. The SAM (Successive Approximation Model) Approach to eLearning - ELM Learning, accessed on April 3, 2025, <https://elmlearning.com/hub/instructional-design/sam-successive-approximation-model/>
18. SAM: A Rapid Design And Development Model - eLearning Industry, accessed on April 3, 2025, <https://elearningindustry.com/sam-successive-approximation-model-for-rapid-instructional-design>
19. 7 Tips To Implement The SAM Model In eLearning, accessed on April 3, 2025, <https://elearningindustry.com/tips-implement-sam-model-in-elearning>
20. What Is Trunk Based Development: How It Works? - Aviator, accessed on April 3, 2025, <https://www.aviator.co/blog/trunk-based-development/>
21. Custom Software vs Off the Shelf: Which Saves You More Money in the Long Run?, accessed on April 3, 2025, <https://kitrum.com/blog/custom-software-vs-off-the-shelf/>
22. Initial cost vs. lifetime cost of software development | LLlnformatics - LLInformatics, accessed on April 3, 2025, <https://www.llinformatics.com/blog/initial-cost-vs-lifetime-cost-of-software-development>
23. Should You Build or Buy Software? AI Might Change Your Answer - FormAssembly, accessed on April 3, 2025, <https://www.formassembly.com/blog/build-or-buy-software/>
24. Definition of elegant program | PCMag, accessed on April 3, 2025, <https://www.pcmag.com/encyclopedia/term/elegant-program>
25. What Makes a Program Elegant? - Communications of the ACM, accessed on April 3, 2025, <https://cacm.acm.org/blogcacm/what-makes-a-program-elegant/>
26. Mastering Elegant Code Part 2: 6 Techniques for Writing Elegant Code - Security Journey, accessed on April 3, 2025, <https://www.securityjourney.com/post/mastering-elegant-code-part-2-6-techniques-for-writing-elegant-code>
27. Software Elegance - Jenkov.com, accessed on April 3, 2025, <https://jenkov.com/tutorials/software-elegance/index.html>
28. Culture & behaviours - Rolls-Royce Career, accessed on April 3, 2025, <https://careers.rolls-royce.com/usa/what-we-offer/culture-and-behaviours>
29. Inspiring Greatness - Rolls-Royce, accessed on April 3, 2025, <https://www.rolls-roycemotorcars.com/en_GB/inspiring-greatness.html>
30. Values - Rolls-Royce, accessed on April 3, 2025, <https://www.rolls-roycemotorcars.com/en_GB/inspiring-greatness/values.html>
31. Our Culture & Values - BMW Group Careers, accessed on April 3, 2025, <https://www.bmwgroup.jobs/br/en/culture.html>
32. Time Consciousness: Eternal Now: Exploring the Concept of Eternal Now in Modern Philosophy - FasterCapital, accessed on April 3, 2025, <https://fastercapital.com/content/Time-Consciousness--Eternal-Now---Exploring-the-Concept-of-Eternal-Now-in-Modern-Philosophy.html>
33. The eternal now - Alan Watts' Philosophy on Authentic Living - Medium, accessed on April 3, 2025, <https://medium.com/@matthewandersonnel/embracing-the-eternal-now-insights-from-alan-watts-a15f9e8503a7>
34. Time & Experience: Twins of the Eternal Now? - ResearchGate, accessed on April 3, 2025, <https://www.researchgate.net/publication/279506213_Time_Experience_Twins_of_the_Eternal_Now>
35. WALKING IN THE ETERNAL NOW | PDF - SlideShare, accessed on April 3, 2025, <https://www.slideshare.net/slideshow/walking-in-the-eternal-now/14917656>
36. State Machines Vs Reactive Programming | Restackio, accessed on April 3, 2025, <https://www.restack.io/p/state-machines-vs-reactive-programming-answer-cat-ai>
37. Reactive programming vs. reactive systems - Akka, accessed on April 3, 2025, <https://akka.io/blog/reactive-programming-versus-reactive-systems>
38. What Is Reactive Programming? | Built In, accessed on April 3, 2025, <https://builtin.com/articles/reactive-programming>
39. The Benefits of Event-Driven Architecture - PubNub, accessed on April 3, 2025, <https://www.pubnub.com/blog/the-benefits-of-event-driven-architecture/>
40. Event-Driven Architecture - AWS, accessed on April 3, 2025, <https://aws.amazon.com/event-driven-architecture/>
41. Event-Driven Architecture (EDA): A Complete Introduction - Confluent, accessed on April 3, 2025, <https://www.confluent.io/learn/event-driven-architecture/>
42. The Complete Guide to Event-Driven Architecture - Solace, accessed on April 3, 2025, <https://solace.com/what-is-event-driven-architecture/>
43. Composition Over Inheritance: Building Flexible and Adaptable Object-Oriented Designs | by Teni Gada | Medium, accessed on April 3, 2025, <https://medium.com/@tenigada/composition-over-inheritance-building-flexible-and-adaptable-object-oriented-designs-8b8966ec4193>
44. Constraint-Based Object-Oriented Programming - Prezi, accessed on April 3, 2025, <https://prezi.com/p/vxujez69h-zg/constraint-based-object-oriented-programming/>
45. Procedural vs Object-Oriented Programming: Understanding the Key Differences, accessed on April 3, 2025, <https://algocademy.com/blog/procedural-vs-object-oriented-programming-understanding-the-key-differences/>
46. Integrating Constraints with an Object-Oriented Language, accessed on April 3, 2025, <https://constraints.cs.washington.edu/cip/kaleidoscope-ecoop-92.pdf>
47. Immutability in Functional Programming - Ada Beat, accessed on April 3, 2025, <https://adabeat.com/fp/immutability-in-functional-programming/>
48. What are the advantages and disadvantages of immutable languages?, accessed on April 3, 2025, <https://langdev.stackexchange.com/questions/444/what-are-the-advantages-and-disadvantages-of-immutable-languages>
49. Immutability – Practical Functional Programming | Part 3 - Sandro Maglione, accessed on April 3, 2025, <https://www.sandromaglione.com/articles/immutability-practical-functional-programming-part-3>
50. Trying to understand the benefits of immutability in imperative programming, accessed on April 3, 2025, <https://softwareengineering.stackexchange.com/questions/430112/trying-to-understand-the-benefits-of-immutability-in-imperative-programming>
51. Systems theory - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_theory>
52. Systems Theory - Sedona Sky Academy, accessed on April 2, 2025, <https://www.sedonasky.org/blog/systems-theory>
53. Systems theory | Social Dynamics, Complexity & Interdependence | Britannica, accessed on April 2, 2025, <https://www.britannica.com/topic/systems-theory>
54. www.britannica.com, accessed on April 2, 2025, <https://www.britannica.com/topic/systems-theory#:~:text=systems%20theory%2C%20in%20social%20science,history%20in%20the%20social%20sciences.>
55. Systems Theory - Sage Publishing, accessed on April 2, 2025, <https://us.sagepub.com/sites/default/files/upm-binaries/32947_Chapter1.pdf>
56. en.wikipedia.org, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_theory#:~:text=Ludwig%20von%20Bertalanffy%20began%20developing,%3A%20Foundations%2C%20Development%2C%20Applications.>
57. Ludwig von Bertalanffy, General System Theory (1968) - Panarchy.org, accessed on April 2, 2025, <https://www.panarchy.org/vonbertalanffy/systems.1968.html>
58. Tektology - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Tektology>
59. en.wikipedia.org, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_theory#:~:text=In%20the%20most%20general%20sense,%22elements%20in%20standing%20relationship.%22>
60. Systems Theory - The Social Work Graduate, accessed on April 2, 2025, <https://www.thesocialworkgraduate.com/post/systems-theory>
61. Systems Theory, accessed on April 2, 2025, <https://www.siue.edu/~adheil/Systems%20Theory%20Paper.pdf>
62. Interdependence - The Personal MBA, accessed on April 2, 2025, <https://personalmba.com/interdependence/>
63. Understanding Structural Family Systems Theory & Therapy - My People Patterns, accessed on April 2, 2025, <https://www.mypeoplepatterns.com/structural-family-systems-theory>
64. Family Systems Theory, accessed on April 2, 2025, <https://web.pdx.edu/~cbcm/CFS410U/FamilySystemsTheory.pdf>
65. Systems Theories - Structural Learning, accessed on April 2, 2025, <https://www.structural-learning.com/post/systems-theories>
66. Systems Theory - The Decision Lab, accessed on April 2, 2025, <https://thedecisionlab.com/reference-guide/psychology/systems-theory>
67. Feedback loops in system thinking | by Myroslava Zelenska - Medium, accessed on April 2, 2025, <https://medium.com/@myroslavazel/feedback-loops-in-system-thinking-7ef06e2ff310>
68. Emergent Properties - Stanford Encyclopedia of Philosophy, accessed on April 2, 2025, <https://plato.stanford.edu/entries/properties-emergent/>
69. Emergence: The Key to Understanding Complex Systems - Systems Thinking Alliance, accessed on April 2, 2025, <https://systemsthinkingalliance.org/the-crucial-role-of-emergence-in-systems-thinking/>
70. Emergent Behavior — Theory - Daposto - Medium, accessed on April 2, 2025, <https://daposto.medium.com/emergent-behavior-theory-a58ef44c0cf0>
71. Emergence - SEBoK, accessed on April 2, 2025, <https://sebokwiki.org/wiki/Emergence>
72. The Role of Emergence in Service Systems, accessed on April 2, 2025, <https://www.sdlogic.net/pdf/post2018/20-Polese-Sarno-Vargo-2020-HICCS.pdf>
73. systemsthinkingalliance.org, accessed on April 2, 2025, <https://systemsthinkingalliance.org/the-crucial-role-of-emergence-in-systems-thinking/#:~:text=Emergent%20properties%20are%20often%20described,behaviour%20of%20individual%20parts%20alone.&text=The%20key%20characteristics%20of%20emergence,wholeness%2C%20dynamic%20and%20downward%20causation.>
74. Systems biology - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_biology>
75. Systems Biology – Old Concepts, New Science, New Challenges - PMC, accessed on April 2, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC3065319/>
76. Perspective: Systems biology beyond biology - Frontiers, accessed on April 2, 2025, <https://www.frontiersin.org/journals/systems-biology/articles/10.3389/fsysb.2022.987135/full>
77. The Historical Evolution of Modern Systems Biology - Clemson University, accessed on April 2, 2025, <https://people.computing.clemson.edu/~steve/Papers/FunSys/prelimdraft1.doc>
78. A Brief History of Systems Biology: “Every object that biology studies is a system of systems.” Francois Jacob (1974) - PubMed Central, accessed on April 2, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC1626627/>
79. An Overview of the Systems Engineering Knowledge Domain - MIT, accessed on April 2, 2025, <http://web.mit.edu/esd.83/www/notebook/sysengkd.pdf>
80. Systems engineering - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_engineering>
81. A Brief History of Systems Engineering - SEBoK, accessed on April 2, 2025, <https://sebokwiki.org/wiki/A_Brief_History_of_Systems_Engineering>
82. (Brief) History of Systems Engineering - INCOSE, accessed on April 2, 2025, <https://www.incose.org/about-systems-engineering/history-of-systems-engineering>
83. www.incose.org, accessed on April 2, 2025, <https://www.incose.org/about-systems-engineering/history-of-systems-engineering#:~:text=The%20term%20systems%20engineering%20dates,engineering%20during%20World%20War%20II.>
84. About Systems Engineering - INCOSE, accessed on April 2, 2025, <https://www.incose.org/about-systems-engineering>
85. Introduction to systems theory in social work - Online MSW Programs, accessed on April 2, 2025, <https://www.onlinemswprograms.com/social-work/theories/systems-theory-social-work/>
86. Systems theory in anthropology - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Systems_theory_in_anthropology>
87. www.rep.routledge.com, accessed on April 2, 2025, <https://www.rep.routledge.com/articles/thematic/systems-theory-in-social-science/v-1#:~:text=Systems%20theory%20in%20the%20first,Gabriel%20Almond%20in%20political%20science.>
88. Ludwig von Bertalanffy - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Ludwig_von_Bertalanffy>
89. Ludwig von Bertalanffy: Man Behind General Systems Theory - The Systems Thinking, accessed on April 2, 2025, <https://thesystemsthinking.com/ludwig-von-bertalanffy-exploring-the-world-through-general-systems-theory/>
90. Improving Healthcare by Embracing Systems Theory - PMC - PubMed Central, accessed on April 2, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC4947551/>
91. thesystemsthinking.com, accessed on April 2, 2025, <https://thesystemsthinking.com/ludwig-von-bertalanffy-exploring-the-world-through-general-systems-theory/#:~:text=Ludwig%20von%20Bertalanffy%20developed%20General,can%20affect%20the%20whole%20system.>
92. Cybernetics - MIT, accessed on April 2, 2025, <https://web.mit.edu/esd.83/www/notebook/Cybernetics.PDF>
93. From Cybernetics to AI: the pioneering work of Norbert Wiener - Max Planck Neuroscience, accessed on April 2, 2025, <https://maxplanckneuroscience.org/from-cybernetics-to-ai-the-pioneering-work-of-norbert-wiener/>
94. Norbert Wiener, the Pioneer of Cybernetics - IT Museum DataArt, accessed on April 2, 2025, <https://museum.dataart.com/short-stories/norbert-wiener>
95. Norbert Wiener: The Father of Cybernetics - The Systems Thinking, accessed on April 2, 2025, <https://thesystemsthinking.com/norbert-wiener-the-father-of-cybernetics/>
96. A Brief History of Systems Thinking, accessed on April 2, 2025, <https://systemsthinkingalliance.org/brief-history-of-systems-thinking/>
97. System Theory in International Relations: Alexander Bogdanov's Tektology Potential, accessed on April 2, 2025, <https://journals.eco-vector.com/0869-0499/article/view/676204>
98. Aleksandr Bogdanov and Systems Theory - PhilArchive, accessed on April 2, 2025, <https://philarchive.org/archive/GARABA-3>
99. Alexander Bogdanov - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Alexander_Bogdanov>
100. en.wikipedia.org, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Tektology#:~:text=According%20to%20Bogdanov%20%22the%20aim,considered%20that%20any%20complex%20should>
101. DOE Explains...Systems Biology - Department of Energy, accessed on April 2, 2025, <https://www.energy.gov/science/doe-explainssystems-biology>
102. Biological system - Wikipedia, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Biological_system>
103. System - Definition and Examples - Biology Online Dictionary, accessed on April 2, 2025, <https://www.biologyonline.com/dictionary/system>
104. Biological Systems | List of High Impact Articles | 587, accessed on April 2, 2025, <https://www.iomcworld.com/scholarly/biological-systems-journals-articles-ppts-list-587.html>
105. Human body systems: Overview, anatomy, functions - Kenhub, accessed on April 2, 2025, <https://www.kenhub.com/en/library/anatomy/human-body-systems>
106. en.wikipedia.org, accessed on April 2, 2025, <https://en.wikipedia.org/wiki/Biological_system#:~:text=A%20biological%20system%20is%20a,scale%20are%20populations%20of%20organisms.>
107. accessed on December 31, 1969, <https://github.com/heymumford/Samstraumr/blob/main/Proposals%20Documentation.md>
108. accessed on December 31, 1969, <https://github.com/heymumford/Samstraumr/blob/main/SamstraumrTestingStrategy.md>