***Title:***

*Keeping Architectures Relevant: Using Domain-Driven Design and Emergent Architecture to Manage Complexity and Enable Change.*

***Abstract:***

Many systems end up becoming legacy before their time. Regardless of the design effort early in the system’s lifecycle, over time the initial architecture becomes progressively less relevant as the codebase changes and matures. The accidental complexity of the system rises as quality of the codebase degenerates, making changes and new features more difficult to accommodate. While the essential complexity, is neglected.

Continuous, considered, Intentional/calculated resolution of significant concerns. Recognizing that the domain model as expressed in our systems is a valuable, core business asset. Domain-driven design involves expending effort on defining and refining the domain model and the establishment of a ubiquitous language--which removes the built-in translation layer between domain experts and the development team--is key to relevant modeling This paper focuses on the key practices that need to be in place for such an effort to succeed.

Domain-driven design enables the architect to articulate the architecture through models that are both relevant to the business and clear to the development team.

Architects must allow architectures to mature, emerge and evolve, fully expecting that their initial design will both morph and become more useful to domain experts and the development team over time

***Reader’s Takeaways***

1. The establishment of a ubiquitous language, which removes the built-in translation layer between domain experts and the development team, is key to relevant modeling.
2. Domain-driven design enables the architect to articulate the architecture through models that are both relevant to the business and clear to the development team.
3. Architects must allow architectures to mature, emerge and evolve, fully expecting that their initial design will both morph and become more useful to domain experts and the development team over time.

***Author Bio’s***

**Brandon Satrom**

Bio here…

**Paul Rayner**

Paul Rayner is a Denver-based independent consultant with more than twenty years of software development and consulting experience. His company, Virtual Genius LLC, helps organizations implement well-crafted enterprise software solutions using agile development principles and practices. Being a strong advocate for software ecology and open source development in .NET, Paul is passionate about software craftsmanship and lean software architecture - teaching others through public courses, coaching, speaking and writing. He has worked in a wide range of industries, including Government, Education, Mining, Insurance, Financial Services and Public Health.

Paul is the founder and president of the Denver chapter of the International Association of Software Architects (IASA), and president of the IASA Chapter Leadership Board. He is an activist for innovation and improvement in the agile, .NET and IT architect communities in Colorado, and has presented at DOSUG, CSOSS, the North Colorado .NET Users Group, IASA Denver and Agile 2009. Paul holds graduate degrees in computing science, theology and philosophy. He writes with an Australian accent about software development at [www.virtual-genius.com](http://www.virtual-genius.com) and about the intersection of faith and work at [www.rayneronline.com/blog](http://www.rayneronline.com/blog).

***List of Previously Published Articles***

Rayner, P. (1990) *A Delaunay-based technique for the manipulation of triangulated surfaces*. Proc. AURISA 90: 483-492. Aust. Urban and Regional Information Systems Assoc.: Canberra, Australia.

Rayner, P. (1990) *The Delaunay Pyramid – Implementing a New Surface Representation Technique* (Honours Thesis). Curtin University of Technology: Perth, Australia.

Rayner, P. (2009). *Book Review - The Problem of Evil: The Gifford Lectures Delivered in the University of St. Andrews in 2003.* The Denver Journal: Denver, USA. Volume 12.