Topic: Evolutionary Software Architectures

presenter: Craig Lawrence

Many of Concepts from DevNexus 2016 in Atlanta

Nothing Here is Rocket Science

- Simple Concepts
- Few New Ideas
- Few New Understandings
- Nothing here is set in stone or meant to be taken out of context (example agile customer)

Role of Architecture in Software Development

- Provides a strong foundation for product
- Avoids the big bungled mess
- Supports a quality product
- Adapts for future change
- Influences team skill sets and structure
- Allows adaptation of new technologies
- Means making right choices rather than blindly following trend of the moment

Traditional Software Approach and Trends

- Waterfall moving to agile
- Monolithic architecture moving to components / micro-services
- Functional teams moving to cross functional teams
- Long and complex product release cycles to shorter more frequent releases
- Long delays in deploying new features
- Change is difficult, painful, and expensive
- User interface experts, prototyping, and user focus groups used to define functionality
- Rigid and hard to adapt to new disruptive technologies

What is an Evolutionary Software Architecture?

“Evolutionary” according to Merriam Webster:

- a process of change in a certain direction
- a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state
- the process by which changes in plants and animals happen over time

Darwin’s Theory of Evolution:

The theory of evolution by natural selection, first formulated in Darwin's book "On the Origin of Species" in 1859, is the process by which organisms change over time as a result of changes in heritable physical or behavioral traits

Darwin's Theory of Evolution is a slow gradual process. Darwin wrote, "… Natural selection acts only by taking advantage of slight successive variations; she can never take a great and sudden leap, but must advance by short and sure, though slow steps.”

"life is a struggle against entropy”

Entropy:

lack of order or predictability; gradual decline into disorder

synonyms: deterioration, degeneration, crumbling, decline, degradation, decomposition, breaking down, collapse; disorder, chaos

“Just as the constant increase of entropy is the basic law of the universe, so it is the basic law of life to be ever more highly structured and to struggle against entropy” Vaclav Havel

“The goal of an evolutionary software architecture is to create an architecture that supports the creation and maintenance of software so that it supports a process of continuous change from a worse state to a higher or better state” (my definition)

What is necessary to adopt a successful Evolutionary Software Approach?

Evolutionary Software Approach: Foundation

- Agile Development Teams
- Component Based Deployment (any language, deploy individually)
- Automated Fitness Function to enforce metrics & test constraints

Evolutionary Software Approach: Effectiveness

- Architecture & Team aligned (avoid pipelining teams)
- Continuous Delivery (automated gradual launch with auto rollback mechanism)
- Feature Toggles (simplify introduction of new or temporary functionality)
- Deployment Pipeline (build & verify)
- Cross Functional Teams (concept to production support all one team)
- Common Architectural Principals
- Regular Architectural Briefings

Evolutionary Software Approach: Practices

- Bring Pain Forward (forces learning & improvement)
- Recognize Last Responsible Moment (moment when complexity or cost will increase significantly)
- Hypothesis (A/B testing) to evolve & discover what best improves the product

Traditional Architecture vs Evolutionary Architecture Comparision:

Change is rare event vs Development team changes production system every day

Fork lift upgrade vs gradual scale up process with rollback (example Netflix)

Collecting logs to discover issue vs Designing experiment to discover solutions (Facebook example)

DB, DEV, QA, SI, CS teams vs single team from concept to production with narrow scope of one component

Upgrade scripts system wide vs running multiple components that stay up are automatically removed only when no longer used

Failure handling on error vs acceptable failure adaptive behavior (default catalog rather than fail when custom catalog not available)

Failure testing vs force failures daily to validate system (Netflix example)

Conclusions

- Big Monolithic architectures for systems are going away
- Software systems are evolving using software that adapts, experiments, and learns what is the better solution
- Complex systems can constantly try different solutions and approaches to better solve problems
- This approach requires making good architectural decisions to allow the software to evolve, adapt, learn, & grow.
- A successful architecture is more than component blocks on a page. It impacts the product development process and decisions in many ways.

Of Interest:
http://www.radiolab.org/story/trust-engineers/

Off topic that may be of interest:
http://www.radiolab.org/story/darwins-stickers/
http://www.radiolab.org/story/188858-krulwich-wonders-right-be-forgotten-biggest-threat-free-speech-internet/
http://www.radiolab.org/story/137407-talking-to-machines/

Institutional Imperative: (consider does it apply to software development?)
http://www.travismorien.com/FAQ/shares/wbinstitutionalimperative.htm