Architectural, Development Lifecycle, and Programmatic Considerations of Hyperexponential Change

John Georgas
The Aerospace Corporation
Institute for Software Research (UCI)
Outline

- **Software Architecture**
- Hyperexponential Change
- Raging Incrementalism
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Software Architecture

- Structures of architectural elements.
  - Components (and their parameters).
  - Interconnections (preferably explicitly).

- Software architecture promises systems that are easier to understand, build, maintain, and analyze.

- Software architecture is about *composition*.
  - Composing elements to achieve system goals.
  - Focus on *architecting*, rather than *building*.
  - High-level view of systems; not getting mired in the low-level implementation details.
System Composition

- Not all architectures are created equal.
  - *When* and *how* are components bound?
    - Early and late binding.
    - Static and dynamic binding.
  - Often, these issues are addressed by *architectural styles*.
  - *Late* and *dynamic* binding are ideal.

- Just some examples:
  - One component subsumes another.
  - Components share a common data store.
  - Remote components exchange data.
Outline

- Software Architecture
- **Hyperexponential Change**
- Raging Incrementalism
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Accelerating Change

- Examine the rate of change of change.
- Trends show:
  - Exponential improvement in rate of change and doubling periods.
  - By 2025, 100 years of 90’s type progress.
  - By 2101, 20000 years of 90’s type progress.
- The result is *hyperexponential change*.
- Everything changes…
  - “Everything you knew yesterday is wrong today.”
  - The system you’ve just designed is already obsolete.
Architectural Requirements

- *Late binding* of components.
  - As far away from source code composition as possible.

- *Dynamic binding* of components.
  - Bindings that are changeable at runtime.

- *Embracing change* through appropriate architectural methodologies and techniques.
Outline

- Software Architecture
- Hyperexponential Change
- **Raging Incrementalism**
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Raging Incrementalism

- System engineering based on:
  - Components (*bricks*) made of:
    - Commodity hardware piece-parts.
    - Open-source software.
  - Interconnections through:
    - Protocols-based interaction.
    - REpresentational State Transfer (REST) architectural style.
Outline

- Software Architecture
- Hyperexponential Change
- Raging Incrementalism
- **Raging Incremental Development**
  - **Architectural Principles**
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Architectural Principles

- Naturalistic architectural design.
  - Based on “naturally occurring” materials.
  - Architecture “grown” from these materials.
  - *Architect* around the building blocks, rather than building around the architecture.
  - Continually evolving entity.

- Explicit architectural model description.

- Based on REST architectural style.
  - Independent components.
  - Stateless interactions.
  - Components modeled as *resources* with their own *namespace* (which may expand).
  - Late and dynamic binding.
    - Interconnections using URI addressing.
Outline

- Software Architecture
- Hyperexponential Change
- Raging Incrementalism
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Lifecycle Considerations
Rapid System Engineering

- **Elicit Requirements**
  - System requirements, as understood at the time.

- **Evaluate Commodity Hardware**
  - Cheap, non-specialized, easily replaced, hyperexponentially improving.

- **Evaluate Open-Source Software**
  - *Free* (in more ways than one), diverse.

- **Architect**
  - Architecture may be refined or revised due to:
    - Increased requirements understanding.
    - New hardware/software components.
Lifecycle Considerations

1. Elicit Requirements
2. Draft Requirements
3. Evaluate Commodity Hardware
4. Refined Architecture
5. Evaluate Open-Source Software
6. Revised Architecture
7. Develop Software Architecture
8. Test Prototype
9. Rapid System Prototyping
10. Rapid System Engineering

Architectural Model

Develop Prototype

Integrate

Evaluate Results

Evaluation Results

Revised Architecture

Software Candidates

Hardware Candidates

Evaluate

Hardware

Test Results
Rapid System Prototyping

- Develop Prototype
  - Rapidly developed.
  - Relative to effort, extremely fully-featured.

- Test Prototype

- Evaluate Results

- Architect
  - May be refined or revised due to:
    - Insight gained during prototyping.
    - Failure to meet goals/requirements during testing.
Outline

- Software Architecture
- Hyperexponential Change
- Raging Incrementalism
- **Raging Incremental Development**
  - Architectural Principles
  - Lifecycle Considerations
  - **Programmatic Considerations**
- Example: Launch Range Video
- Summary
Development Cycles

- Measured in *weeks* and *months*, not years.
  - A few months (at the most) for an initial iteration resulting in a complete prototype.
  - Weeks for further iterations resulting in revisions and refinements.

- Independent and parallel modification of components.
  - Promoted through decoupled RESTful interactions.

- Shorter cycles result in less management overhead.
  - Promote early incorporation of lessons-learned.
Productivity

- Development focused on *integration*, not coding.
  - Conventional productivity measures (LOC, function points) inappropriate in this context.

- Measures and project milestones focused on:
  - Scale of integrated components.
  - Functionality achieved.

- Focus on quality, not quantity.
Staffing

- Exceptional *architects* and *integrators* take precedence over programmers.
  - Development focuses on integration, rather than “in-house” development.

- Special role – *software surveyor*
  - Knowledge of open-source projects – both existing, and planned.
  - Aware of strengths and weaknesses of projects, as well as dependencies and requirements.
  - This knowledge is essential:
    - Formulates early prototype systems.
    - Enables and guides long-term planning.
Outline

- Software Architecture
- Hyperexponential Change
- Raging Incrementalism
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- **Example: Launch Range Video**
- Summary
RAnge Video Experiment

- Experimental replacement for video monitoring at Eastern and Western launch ranges.

- Features:
  - Large-scale, distributed system.
  - Real-time video encoding and decoding.
  - Video archival.
Raging Incremental Prototype

- Peer-to-peer architecture.
- RESTful interactions.
- Developed in a 6 week cycle.

**Video Camera Brick:**
- Shuttle PC with Firewire camera.
- Network camera control.
- Software-based MPEG-4 encoding.
- Precision timestamps for video frames.
- Digital video streaming.

**Video Proxy Brick**
- Custom Python program.
- Live.com streaming libraries.

**Video Distribution Server Brick**
- Based on open-source Darwin Streaming Server from Apple.

**Video Archive Brick**
- 4U commodity rack server containing 4 terabytes of storage.
- FreeBSD.
Components Used

- **Video Camera Brick**
  - **Hardware**
    - Shuttle form-factor box.
    - Unibrain Firewire camera.
  - **Software**
    - Debian Linux OS (open source).
    - libdc1394 (open source).
    - spook broadcaster (open source).
    - xViD MPEG4 encoding (open source).

- **Video Distribution Brick**
  - **Hardware**
    - Shuttle form-factor box.
  - **Software**
    - Debian Linux OS (open source).
    - Darwin Streaming Server (open source).

- **Video Archive Brick**
  - **Hardware**
    - 4U commodity rack server (4 TB).
  - **Software**
    - FreeBSD OS (open source).
    - mencoder MPEG4 encoder (open source).
    - mplayer for video playback (open source).
    - MySQL database (open source).

- **Video Proxy Brick**
  - **Hardware**
    - Shuttle form-factor.
  - **Software**
    - Custom Python video proxy.
    - Live.com RTP/RTSP libraries (open source).
  - **Protocols**
    - RTSP, RTP, HDP, HTTP, TCP.
  - **Standards**
    - IEEE1394.
  - **Languages**
    - C, C++, Python.
Outline

- Software Architecture
- Hyperexponential Change
- Raging Incrementalism
- Raging Incremental Development
  - Architectural Principles
  - Lifecycle Considerations
  - Programmatic Considerations
- Example: Launch Range Video
- Summary
Summary

- Hyperexponential change changes everything.
- Raging incrementalism
  - Architecture must be about composition.
  - Late and dynamic component binding.
- Formulations of architectural principles, development lifecycle, and programmatic insight.
- Proof-of-concept prototype in the RAnge Video Experiment (RAVE) system.
What We Learned…

- Success comes at a price.
  - Tangible results rest on a mountain of failure.
- Going at it alone can hurt.
  - Walk a fine line between asking for help and being self-sufficient.
- Flexible and understanding management is the key to happiness.