Software engineering: Architecture-driven Development

Richard Schmidt
Sirrush Corporation
703–919–8531

NDIA 15th Annual Systems Engineering Conference
Hyatt Regency Mission Bay
San Diego, California
October 24
Overview

- Software Development *CHAOS*
- What is a Software Architecture
- How is a Software Architecture Developed
- Software Engineering Practices
- Software Architecture Design Strategy
- Relationship to Other Software Methodologies
Software Development Trends

Short History of Software Methods, By David F. Rico

Demand for Software Production


Scientific Computing

Flow Charts

Industrial/Business Computing

Structured Design
Cost Estimation
Code Inspections
Structured Analysis
IDEF
Defect Prevention
Object-Oriented Design
Software Reuse
Object-Oriented Analysis
Risk Management
Software Architecture
Software Metrics
CASE Tools
Automated Regression Testing
Integrated Development Environments (IDEs)
Unified Modeling Language
Rapid Application Development
Agile

Personal Computing

Computer-Aided Software Engineering
Non-Software Engineering Methods

Short History of Software Methods, By David F. Rico


(c)2012 SIRRUSH Corporation
Chaos Reports

In the United States, we spend more than $250 billion each year on IT application development of approximately 175,000 projects. The average cost of a development project for a large company is $2,322,000; for a medium company, it is $1,331,000; and for a small company, it is $434,000. A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys — hear no failures, see no failures, speak no failures.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>16%</td>
<td>27%</td>
<td>26%</td>
<td>28%</td>
<td>34%</td>
<td>29%</td>
<td>35%</td>
<td>32%</td>
</tr>
<tr>
<td>Challenged</td>
<td>53%</td>
<td>33%</td>
<td>46%</td>
<td>49%</td>
<td>51%</td>
<td>53%</td>
<td>46%</td>
<td>44%</td>
</tr>
<tr>
<td>Failed</td>
<td>31%</td>
<td>40%</td>
<td>28%</td>
<td>23%</td>
<td>15%</td>
<td>18%</td>
<td>19%</td>
<td>24%</td>
</tr>
</tbody>
</table>

When a bridge falls down, it is investigated and a report is written on the cause of the failure. This is not so in the computer industry where failures are covered up, ignored, and/or rationalized. As a result, we keep making the same mistakes over and over again.

CHAOS, The Standish Group Report, 1995
Why Such CHAOS?

- Computer technology’s rapid transition into Industrial, Commercial & Consumer systems/products
- Majority of Software R&D
  - Initially Programming Language focused (1950–1985)
  - Programming productivity focused (1985–2010)
- Software development project management emphasis on documentation
  - Inadequate design methodologies
  - Software Professional untrained in “product” design
- Software workforce demand exceeded availability of skilled professionals
- Variety of software application domains
- No sponsored research to establish formal software design practices

(c)2012 SIRRUSH Corporation
Software As a Critical Material

- Software Lawsuits
- Software Architectural Design
- Professional Practices
- Product Stability & Quality
- Time-to-Market & Price Competition
- Pioneering Design Practices
- Software Warranties
- Initial Software Liability Litigation
- High Order Languages
- Software Development Methodologies
  (Project Management Controls)
- Programmatic Design Techniques
- Assembly Language Programming
- Design Heuristics (Trial & Error)
- Flow Charts
- Direct Machine Language
- Design Unnecessary
- Consumer Applications
- Consumer Protection
- Commercial Applications
- Industrial Applications
- R&D/Scientific Computing

Software Development Professionalism
Applying Systems Engineering Practices

“The Government’s present system for procuring software does not meet the Government’s needs and wastes resources. The application of “systems engineering” disciplines is needed to remedy the procurement system’s defects. . . Software Development is a complex process that requires modern “systems engineering” techniques.”

Bugs In The Program, Problems in Federal Government Computer Software Development and Regulation, Staff Study by the Subcommittee on Investigations and Oversight, Congress, September 1989

What are “systems engineering” disciplines?

How can “systems engineering” be adapted to development of software product?
What is Architecture-driven Development?

• Establishing a Requirements Baseline
  – Balance the needs and expectations of all stakeholders
  – Provides a basis for DESIGNING the software product
  – Establishes the basis for software acceptance testing
• Establishment of a comprehensive software product design
  – Functional basis for ensuring product performance
  – Structural basis for software implementation
• Software Post-development Processes Specifications
• Full traceability throughout the software architecture
  – Software Specifications, Functional Specifications, Physical Specifications
• Basis for continual planning and resource allocation
• Architectural Design Decisions
  – Risk-based decision-making
  – Focus on project success criteria
Organizing for Software Engineering

Software Development

Software Engineering IPT
- Computing Environment
- Software Implementation
- Software Test and Evaluation

Post-development Process IPT
- Product Distribution
- Product Training
- Product Support
Value of Software Architecture

Provides specifications for every software module, routine or class

Operational Models

Functional Models

Physical Models

Software Specifications

Functional Specifications

Structural Specifications

Basis for Technical and Project Planning
- Software Breakdown Structure
- Work Packages & Dependencies
- Resource Allocations
- Integrated Technical Planning
- Integrated project Planning

Objective architectural decision-making
- Project objectives
- Resource constraints
- Technical challenges
- Risk aware

Development Speed (Time-to-Market)

Product Cost

Development Program Expense (Investment)

Product Performance
Developing The Software Architecture

- Business Needs and Expectations
- Operational Concepts

Software Requirements Analysis

- Operational Model
- Software Requirement Specifications

Requirements Baseline

Functional Analysis and Allocation

- Functional Behavior Model
- Functional Hierarchy
- Functional Specifications

Software Analysis
  - Trade-off Analysis
  - Risk Assessment
  - Complexity Assessment

Functional Architecture

Software Design Synthesis

Physical Architecture

- Structural Unit Specifications
- Structural Component Specifications
- Component Integration Strategy
Design Decisions

- It’s not the *Decision* that matters – It’s the **Rationale**
- **Decision** implies a choice among multiple alternatives
- The important architectural decisions affect software product life-cycle characteristics:
  - Complexity
  - Supportability
  - Extensibility
  - Usability
  - Product Life-cycle Costs
- Architectural Decisions must align technical scope of work with availability of project resources
Software development is a technical effort
All technical challenges impact project feasibility
All risk to a software development project is technical in nature
- Insufficient resources should imply a less robust software product
- Complexity must be simplified
- Over-stated requirement must be challenged
Software prototyping should be used to assess technical solution feasibility
- Never put a prototype on the CRITICAL PATH
1. Requirements Definition Stage
   - Requirements Analysis – Translates stakeholder needs into software requirements specifications
   - Functional Analysis and Allocation – Analyzes ambiguous needs or requirements to grasp functional and performance characteristics
   - Verification – Confirms that every software requirement can be traced to stakeholder needs or derived from analytical studies

2. Preliminary Architecture Definition Stage
   - Functional Analysis and Allocation – Analyzes specified requirements to derive more detailed functional and performance understanding
   - Software Design Synthesis – Analyzes functional components to confirm an acceptable design solution exists
   - Requirements Analysis – Analyzes functional components and units to specify their behavior and performance characteristics
   - Verification – Confirms that every functional unit and component can be traced to software requirements or derived from analytical studies

3. Detailed Architecture Definition Stage
   - Application Design Synthesis – Combines functional units to compose structural units
   - Requirements Analysis – Integrates and deconflicts functional unit requirements to specify structural units
   - Functional Analysis and Allocation – Analyzes structural units to derive functional integration strategies
   - Software Design Synthesis – Assembles and integrates structural units to compose structural components
   - Requirements Analysis – Integrates and deconflicts structural unit requirements to specify structural components
   - Verification – Confirms that structural units and component specifications align with the functional architecture
   - Validation – Confirms that every structural unit and component can be traced to specified software requirements

Software Engineering Practices
Planning the Software Engineering Effort

Perform Software Engineering Activities

Initial Planning
- Focus on Software Requirement Definition Phase
- Identify Milestones Success Criteria

Technical Planning (Revision 1.0)
(Focus on Preliminary Architecture Definition Phase)
- Software Engineering Plan
- Software Implementation Plan
- Software Test Plan
- PDSS Process Development Plan(s)

Technical Planning (Revision 2.0)
(Focus on Detailed Architecture Definition Phase)
- Software Engineering Plan
- Software Implementation Plan
- Software Test Plan
- PDSS Process Development Plan(s)

Technical Planning (Revision 3.0)
(Focus on Software Implementation Phase)
- Software Engineering Plan
- Software Implementation Plan
- Software Test Plan
- PDSS Process Development Plan(s)

Integrated Master Plan and Revisions

Integrated Master Schedule and Revisions

Integrated Plan and Revisions

3 Primary Planning Iterations
1. Preliminary Architecture Definition Phase
2. Detailed Architecture Definition Phase
3. Software Implementation Phase

Project Plan and Revisions

Software Requirement Baseline

Product Functional Architecture

Product Physical Architecture
Elements of Software Architecture

Stakeholder Needs and Expectations

Requirements Baseline
- Software Requirements Specification(s)
- Software Interface Specification(s)
- PDSS Process Specifications

Functional Architecture
- Functional Decomposition
- Performance Measure Allocation among Functions
- Control Logic
- Data Flows (Functional Interfaces)
- Failure Detection and Recovery
- Resource Utilization
- Behavior Models
- Design Diagrams & Documentation
- Functional Specifications
- Data Definitions

Physical Architecture
- Structural Units (Building Blocks)
- Structural Component Integration Strategy
- Structural Unit Specifications
- Data Specifications
- Structural Component Specifications
- Component Integration Test Objectives

Computing Environment
- Data Processing Platforms
- Networking / Communication Infrastructure
- Data Storage & Replication
- Concurrent User Interaction
- Administration Control Station

Software Product Architecture

Post Development Processes
- Training Operations Plan
- Training Material
- Training Environment
- Training Procedures
- Distribution Operations Plan
- Distribution Material
- Distribution Environment
- Distribution Procedures
- Product Support Operations Plan
- Product Support Process
- Problem Resolution
- Enhancements
- Iterations of Development Process
- Product Support Environment
- Customer Support Operations Plan
- Customer Support Material
- Customer Support Environment
- Customer Support Procedures

Test & Evaluation
- Test Plan
- Test Procedures
- Test Environment
Artifacts of Software Architecture

Software Architecture

Software Product Architecture

- Requirements Baseline
  - S/W Requirement Specification
  - S/W Interface Specification
  - Requirement Traceability Matrix
  - Entity Relationship Diagram

- Functional Architecture
  - Behavior Model
  - Execution Timeline
  - Functional Decomposition Diagram
  - Entity Relationship Diagram
  - Functional Component Specifications
  - Functional Unit Specifications

- Physical Architecture
  - Interface Block Diagram
  - Structural Configuration Diagram
  - Engineering Assembly Diagram
  - Software Integration Diagram
  - Entity Relationship Diagram
  - Structural Component Specification
  - Structural Unit Specification

Post-Development Process Architecture

- Distribution Process Architecture
- Training Process Architecture
- Sustainment Process Architecture
- Customer Support Process Architecture
- Product Support Process Architecture
Software Architecture Definition

<table>
<thead>
<tr>
<th>Preliminary Architecture Definition</th>
<th>Detailed Architecture Definition</th>
</tr>
</thead>
</table>

**PDR**

- Preliminary Functional Architecture
- Initial Structural Design Concept
- Preliminary Test Procedures
- Updated Requirement Traceability Matrix

**CDR**

- Revised Functional Architecture
- Completed Physical Architecture
- Initial GUI Structural Design
- Detailed Test Procedures
- Updated Requirement Traceability Matrix

**Products of Preliminary Architecture Definition:**
- Behavior Model (Data Processing Transactions)
- Functional Hierarchy
- Conceptual Design Structure
- Functional Component Specifications
- Functional Unit Specifications
- Database Transaction Specifications
- Interface Transaction Specifications (Protocols & Messaging)
- User Interface Functional Hierarchy

**Products of Critical Architecture Definition:**
- Structural Unit Specifications
- Software Integration Strategy
- Structural Component Specifications (Integrated Behaviors)
- Interface Design Documents
- Structural Block Diagrams
- Structural Interface
- User Interface Structural Hierarchy
### Software Architecture Definition

<table>
<thead>
<tr>
<th>Preliminary Architecture Definition</th>
<th>Detailed Architecture Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish resource utilization</td>
<td>Establish resource utilization</td>
</tr>
<tr>
<td>functional specifications:</td>
<td>structural specifications:</td>
</tr>
<tr>
<td>• Allocate resources among functions</td>
<td>• Behavioral thread profiles</td>
</tr>
<tr>
<td>• Identify resource supervision behaviors</td>
<td>• Structural component specification</td>
</tr>
<tr>
<td>• Structural unit specification (if desired)</td>
<td>• Implement resource utilization stub specifications</td>
</tr>
<tr>
<td>Establish the computing resource utilization strategy</td>
<td>Implement resource utilization requirements:</td>
</tr>
<tr>
<td>• Software design and coding guidelines</td>
<td>• Design units with efficient object creation &amp; destruction mechanisms</td>
</tr>
<tr>
<td>• Identify task prioritization strategy</td>
<td>• Implement connection &amp; object pools</td>
</tr>
<tr>
<td>• Identify multi-tasking scheduling strategy</td>
<td>Measure computing resource utilization</td>
</tr>
<tr>
<td>• Identify garbage collection strategy</td>
<td>• Software unit resource consumption and conservation (average &amp; worst-case)</td>
</tr>
<tr>
<td>• Identify resource queuing strategy</td>
<td>• Integrated component resource consumption and conservation</td>
</tr>
</tbody>
</table>

### Software Implementation

<table>
<thead>
<tr>
<th>Unit Implementation</th>
<th>Component Integration &amp; Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement resource utilization requirements:</td>
<td>Assess resource utilization:</td>
</tr>
<tr>
<td>• Design units with efficient object creation &amp; destruction mechanisms</td>
<td>• Design units with efficient object creation &amp; destruction mechanisms</td>
</tr>
<tr>
<td>• Implement connection &amp; object pools</td>
<td>• Implement connection &amp; object pools</td>
</tr>
</tbody>
</table>

---

Software performance is predicated on the performance of the Computing Environment
**Software Requirements Definition**
- Operational Model
- Software Requirement Specs
- Interface Requirement Specs
- Software Test Plan
- Software Requirement Baseline

**Project Management**
- Objectives and Constraints
- Plans and Schedules
- Budgets
- Stakeholder Needs & Expectations
- Change Proposals

**Project Control**
- Objectives and Constraints
- Plans and Schedules
- Budgets
- Stakeholder Needs & Expectations
- Change Proposals

**Software Acceptance Testing**
- Test Conduct
- Software Test Report
- Software Problem Reports
- Waivers and Deviations

**Software Implementation (Agile Integration & Testing)**
- Software Component Integration
- Software Component Testing
- Dry-run Acceptance testing

**Software Implementation (Agile Unit Development)**
- Software Unit Design (Programmatic)
- Software Unit Coding
- Software Unit Testing
- Software Development Folders
Summary

• Software Industry is in CHAOS
• Computer technology’s rapid growth and employment have prevented software engineering practices from arising
• Software Product Complexity must be corralled
• Application of Systems Engineering Practices is a viable solution
• Architecture-driven approach improves Software Development Probability of Project Success
• Software Functional Decomposition must be complete to enable a bottom-up structural design solution
• Software Methodologies rely on Programmatic Design & Coding (Prototyping) – Hence CHAOS!
• Software Engineering is the little brother of Systems Engineering
  – Software as a “material” offers unique challenges!