Design, Development, and Testing of PHM Software

Christopher Teubert, Software Engineer/Group Lead
Diagnostics and Prognostics Group
Who Am I

Christopher Teubert

2007-2012  B.S. Aerospace Engineering from Iowa State University (Go Cyclones!)

2010-2012  Asteroid Deflection Research Center (Ames, IA)

2012-2016  Contractor with SGT, Inc. at NASA Ames Research Center’s Diagnostics and Prognostics Group

2015-2019  M.S. Computer Science and Engineering from Santa Clara University (In Progress)

2016-      NASA Ames- Group Lead for Diagnostics and Prognostics Group

Generic Software Architecture for Prognostics (GSAP)  Prognostics As-A-Service (PaaS)
Setting Expectations

Goals:
- Discuss the challenges and processes of creating quality PHM software
- Software Engineering and Systems Engineering

Not covered:
- Algorithm design (See other tutorials/Short course)
- Model-Based System Engineering
- Managing people

Please ask questions throughout
Agenda

1. Problem Definition
2. Project Planning
3. Software Design
4. Software Development
5. Software Testing
6. Final Thoughts
Re-occurring themes

1. Document
2. Do not skip the planning!
3. Traceability
4. Scale process to project size
5. Don’t Duplicate Information
   a. Have a primary location and reference it (avoid reference trees)
6. Automate whenever possible
7. Decomposition

Credit Barry Boehm (1981)
Creating software has common activities. These are done in “phases”, phases **END** is a specific order.

Each phase has:

1. Inputs
2. Objectives
3. Outputs
“Activities” / “Phases”

1. Analysis
   a. Defining the problem (Requirement Definition)
   b. Project planning
2. Design
3. Development
4. Test
5. Maintenance
6. Decommissioning

Analyze -> Design -> Build -> Test
“Activities” / “Phases”

1. Analysis
   a. Defining the problem (Requirement Definition)
   b. Project planning
2. Design
3. Development
4. Test
5. Maintenance - Most often forgotten in project planning
6. Decommissioning
“Activities” / “Phases”

1. Analysis - Where most problems start
   a. Defining the problem (Requirement Definition)
   b. Project planning
2. Design
3. Development
4. Test
5. Maintenance - Most often forgotten in project planning
6. Decommissioning
Defining the Problem
Defining the Problem

Problem Definition

- Initial Customer Expectations
- Stakeholder Expectations
- Common Problem Understanding

Project Planning
Software Design
Software Development
Software Testing
Defining the Problem

Goal: Establish and document a clear, shared understanding of the problem between all stakeholders

*Determining “what to build” - Stakeholder expectations*

Creating a Common Vocabulary (Create a Glossary)

Steps:

- Gather Information >> Analyze >> Document >> Review & Get Buy In
How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it

How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed
Gathering Information & Analysis

*Improve your understanding of the problem beyond what the customer has provided you*

Sources:
1) Existing Solutions, 2) Standards, 3) Customer, 4) Stakeholders
Needs, Goals, Objectives (NGO’s)

Needs: What are the stakeholder needs that have prompted this project

  e.g. Ensure safe operations, reduce costs

Goals: What is the high-level goals that your project is trying to fulfill

  e.g. Reduce in-flight failures, and reduce maintenance costs

Objectives: What are some specific objectives to achieve the goals

  e.g. Provide predictions for 99% of battery end of discharge events to operators with enough time to land safely

Problem Definition

Project Planning  >  Software Design  >  Software Development  >  Software Testing
Concept of Operations

- Description of how a product will be integrated and used
- Spurs requirement definition and discussion with stakeholders
- Write in user’s language!
- Get stakeholder input

Problem Definition  Project Planning  Software Design  Software Development  Software Testing
Context diagram

Live Sensor Data
Configuration
Future Loading

Prognoser

State Estimation
Prognostics Results

Problem Definition
Project Planning
Software Design
Software Development
Software Testing
ConOps Diagrams

- Show system in use
- Show interactions

1) Stakeholders
2) How they interact with the system (actions available)
3) Relationship between actions
Software Requirements

- Define Specific, Measurable, Attainable, Relevant, Time-constrained (S.M.A.R.T) objectives to be met by the system
- Multiple layers, grouped into related requirements
- Defined before design
- Will change in later phases, but less as time goes on
- Contract with stakeholders and developers
- Verified before delivery
Problem Definition

Project Planning

Software Design

Software Development

Software Testing
Requirement Definition

ID
Name
Text
Rationale
Testing Method
Testing Description
Traceability

Requirement Definition

ID: A unique identifier for a requirement.
- Does not change after baselining - even when removed
- Used for referencing
- E.g. [PROJECT]-[SYSTEM]-[TYPE]-###

Name
Text
Rationale
Testing Method
Testing Description
Traceability

Problem Definition
Project Planning
Software Design
Software Development
Software Testing
Requirement Definition

ID
Name: Used to refer to the requirement (for humans)
  • E.g. RUL Update Rate

Text
Rationale
Testing Method
Testing Description
Traceability

Problem Definition
Project Planning
Software Design
Software Development
Software Testing
## Requirement Definition

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Text: The actual requirement</th>
<th>Rationale</th>
<th>Testing Method</th>
<th>Testing Description</th>
<th>Traceability</th>
</tr>
</thead>
</table>

- Typically “shall” statements
- [Subject] shall [requirement]
- e.g. GSAP shall update state of health estimates no less than every once every second, while data is being received

### Problem Definition

- Project Planning
- Software Design
- Software Development
- Software Testing
Requirement Definition

ID
Name
Text
Rationale: Describes purpose of requirement, who is interested in it, and its priority

Testing Method
Testing Description
Traceability

Problem Definition: Project Planning → Software Design → Software Development → Software Testing
ID
Name
Text
Rationale

Testing Method:
1) Inspection: use senses, passively observe
2) Analysis: Engineering judgement based on tests of lower systems
3) Demonstration: Actively load, and observe results
4) Test: With instrumentation and measurements under loading

Testing Description
Traceability

http://reqexperts.com
Requirement Definition

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Text</th>
<th>Rationale</th>
<th>Testing Method</th>
<th>Testing Description: More detailed description</th>
</tr>
</thead>
</table>

Traceability

Problem Definition | Project Planning | Software Design | Software Development | Software Testing
Requirement Definition

ID
Name
Text
Rationale
Testing Method
Testing Description: More detailed description

Traceability

Requirements - Tests
Requirements - Code
Requirements - Children/Parent
Requirements - Objectives
Requirements - Stakeholders

Problem Definition
Project Planning
Software Design
Software Development
Software Testing
## Requirement Traceability Matrix

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Test ID 1</th>
<th>Test ID 2</th>
<th>Test ID 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-ID-1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ-ID-2</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>REQ-ID-3</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ-ID-4</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Problem Definition**
- Project Planning
- Software Design
- Software Development
- Software Testing
# Requirement Traceability Matrix

<table>
<thead>
<tr>
<th></th>
<th>STAKEHOLDER 1</th>
<th>STAKEHOLDER 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-ID-1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>REQ-ID-2</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ-ID-3</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ-ID-4</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Example Software Requirements

- Communication of sensor data
  - What data?
  - Rate?
  - Latency?
  - Meta Information?

- Communication of Results
  - What results?
  - How (UI?)

- Storage
  - What, when, where?

- Hardware Constraints

- Control
  - Starting/stopping?
  - Configuration?
  - Other controls?

- Deployment

- Software Testing
- Software Development
- Software Design
- Project Planning
- Problem Definition
Non-Functional Software Requirements
“The -ilities”

Scalability - Amount of load on system
Extensibility - Adding new features
Security - time between incidents
Accessability -
Robustness - time between errors
Maintainability - mean time to patch
Recoverability

Note:
Can also be expressed as “design considerations”
- Include in review checklists

See also: https://en.wikipedia.org/wiki/Non-functional_requirement
Additional PHM Requirement Considerations

- Uncertainty Management: Data, Models, Future Loading, etc...
- History - data permanence
- Consistency in results
- Performance

See also: Requirements Flowdown for Prognostics and Health Management
Example PHM Performance Requirements

Prognostics Horizons- Minimum time before failure that some condition is met

a. The valve prognoser shall predict valve EOL within [BOUNDS] more than [PROGNOSTICS HORIZON] before actual EOL no less than 99% of the time.

b. Can have multiple points with different conditions - decision points

c. E.g. For aircraft have one point for when there is enough time to complete route, another for emergency landing

d. Often choose conservative bounds

See: Metrics for Offline Evaluation of Prognostic Performance
Example PHM Performance Requirements

α-λ- Minimum condition at set time

a. The valve prognoser shall predict valve EOL within [BOUNDS] at [PROGNOSTICS HORIZON] before actual EOL no less than 99% of the time.
b. Can have multiple points with different conditions

See: Metrics for Offline Evaluation of Prognostic Performance
Diagnostics vs Prognostics Requirements

Different metrics
Different results

Diagnostics - Real-time faults
- Stricter availability
- Less prediction latency
- Presentation Requirements: inform quick decisions

Prognostics - Long term degradation
- Accepting to limited latency
- Accepting to some latency
- Higher computation requirements
Requirement Tracking Software

- Excel/Google Sheets
- Jira
- Targeted Tools (e.g. IBM Doors)

http://makingofsoftware.com/resources/list-of-rm-tools
Remember- REVIEW REQUIREMENTS
Project Planning
Project Planning

- Organizational Procedures
- Customer Expectations
- Problem Understanding

Project Plan

Problem Definition
Project Planning
Software Design
Software Development
Software Testing
Project Planning

1. Build team, define responsibilities
2. Define and **COMMUNICATE** “Process”
   a. Choose/document software development model
   b. Define configuration management & quality assurance policies

Note: Often done in parallel with problem definition
**Software Development Model - Three primary model flavors**

1. Waterfall
2. Spiral
3. Agile

Most others are mixtures of these.
Pure Waterfall

- Sequential
- Output of previous “flows into next”
- Next doesn’t start until previous ends

Requirements → Design → Implement → Test → Maintain

Problem Definition → Project Planning → Software Design → Software Development → Software Testing
Realistic Waterfall

- Overlap between stages
- Amount of overlap depends on organization
Waterfall V-Model

- Problem Definition
- Project Planning
- Software Design
- Software Development
- Software Testing
Spiral

- Iterative waterfall
- Scale effort for each activity based on risk
- Each iteration will add detail, improve on past
- Review with each iteration

Credit: Boehm, 1988

Diagram:

1. Determine objectives
2. Identify and resolve risks
3. Development and Test
4. Plan the next iteration

Project Planning

Problem Definition

Software Design

Software Development

Software Testing
Agile

- Many flavors (Kanban, Scrum, etc.)
- Common elements:
  a. Iterative Development
  b. Collaboration between disciplines/teams
  c. Light design/documentation process
  d. Continuous Feedback

See also: http://agilemanifesto.org
## Choosing a Development Process

<table>
<thead>
<tr>
<th>Factors</th>
<th>Waterfall</th>
<th>Spiral</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclear User Requirement</td>
<td>Poor</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Unfamiliar Technology</td>
<td>Poor</td>
<td>Excellent</td>
<td>Poor</td>
</tr>
<tr>
<td>Complex System</td>
<td>Good</td>
<td>Excellent</td>
<td>Poor</td>
</tr>
<tr>
<td>Reliable system</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
</tr>
<tr>
<td>Short Time Schedule</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
</tr>
<tr>
<td>Cost limitation</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
</tr>
<tr>
<td>Visibility of Stakeholders</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Skills limitation</td>
<td>Good</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Documentation</td>
<td>Excellent</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Component reusability</td>
<td>Excellent</td>
<td>OK</td>
<td>Poor</td>
</tr>
</tbody>
</table>

https://melsatar.blog/2012/03/21/choosing-the-right-software-development-life-cycle-model/
Risks

- Document risks throughout with likelihood and mitigation
- Manage as you manage requirements
Configuration Control Processes

A quality control tool

http://nvie.com/img/git-model@2x.png
Documenting Project Planning

System Engineering Management Plan (SEMP)

Software Development Plan

Think about tools

Think about responsibilities

Think about procedures, configuration management

Not necessarily documents- could be wiki (scale)
Keep to the process

Communicate process

Make sure you’re following it!
Software Design
Software Design

- Plan how you are going to meet the requirements
- Develop a common understanding of “what you will build”
- Communicate “what will be built” to developers, testers, stakeholders
- Uncover and resolve issues early- before building the software
Design Principles

- Abstraction: Only reveal relevant information
- Coupling: Interdependence between modules
- Separation of Concerns: focus on one concern at a time
- Decomposition: Break into smaller, more manageable parts or modules
- Software Reuse: Don’t reinvent the wheel unless you have to
- Design Patterns: Well defined solutions to problems
THE PROJECT I INHERITED HAS WEAK CODE. I NEED TO REWRITE IT FROM SCRATCH.

WILL THERE EVER BE AN ENGINEER WHO SAYS, “THAT LAST GUY DID A GREAT JOB. LET’S KEEP ALL OF IT”?

I’M HOPING THE IDIOT YOU HIRE TO REPLACE ME SAYS THAT.
Key issues

- Concurrency
- Control
- Data persistence
- Distribution
- Error handling
- Presentation
- Security

SWEBoK
Architecture Styles & Design Patterns

- Named collection of architectural and design decisions

Architectural Styles
- Layered
- Pipes & Filters
- Client/Server
- Broker

Design Patterns
- Creational: Builder, Factory, Singleton, ...
- Structural: Adapter, Bridge, Decorator,...
- Behavioral: Interpreter, Iterator, Observer, ...

See Design Patterns: Elements of Reusable Object-Oriented Software

Problem Definition  Project Planning  Software Design  Software Development  Software Testing
Standards

How Standards Proliferate:
(See: A/C chargers, character encodings, instant messaging, etc.)

Situation: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone's use cases. Yeah!

Soon:

Situation: There are 15 competing standards.
Onboard vs Remote

On-board: Availability, less risk and delay but at a cost

Remote: Shared resource (cloud), but cost to 1) Security, 2) Latency, and 3) Availability

Prognostics As-A-Service (PaaS)

See Shankar Sankararaman’s talk tomorrow
Common design - Model-View-Controller

Model: backend- performs PHM

Controller: Accepts input (sensor data, future loading, control, etc.)

View: convey’s the results

Often multiple views for each type of user- with common API for accessing information
Effectively communicating results

Information depends on end user:

- Pilot/Operator: “at a glance” critical information
- Maintainer/Manager: Long-term trends
- Scheduler: Mid-term
Documenting Software Design

- Software Design Specification (SDS)
- Software Design Artifacts (UML, SysML)
UML Activity Diagrams
Generic Software Architecture for Prognostics (GSAP)

- C++ Framework to create prognostics applications
- Includes implemented models and algorithms to build upon
- Includes standard interface for prognostics technologies

See: https://github.com/nasa/GSAP/
GSAP- Details

User Layer:

- Prognostics Application Driver (PAD)
- Prognosers
- Communicators
Example

Config
Communicators: ResultComm.cfg, DataComm.cfg
Prognosers: Example.cfg

Example Driver
Data Communicator
Result Communicator
Prognoser

GUI

Data Stream
Results

DataComm.cfg
ResultComm.cfg
Example.cfg

GSAP
Prognostics Framework
Support Library

Compile

Prognostic Application
Example: Model-Based Prognoser

ModelBasedPrognoser

Observer Algorithm: Unscented Kalman Filter, Particle Filter

Predictor Algorithm: Monte Carlo, Latin Hypercube (soon!)

From Paper: Model-based Prognostics with Concurrent Damage Progression Processes
Example

Problem Definition  Project Planning  Software Design  Software Development  Software Testing
Resources

Design Patterns: Elements of Reusable Object-Oriented Software

Prognostics: The Science of Prediction

A Generic Software Architecture for Prognostics (GSAP) - ijPHM Paper/Github

http://www.uml-diagrams.org

Prognostics Model and Algorithm Library
Software Development
Development Stage

To build the desired product, as described in the design and requirements

Notes:
- Track progress
- Minimize complexity
- Anticipate change
- Construct for verification
- Coding standards - organizational and language
- Test as you go (Unit and intermediate integration)
- Document as you go
- Reuse when possible
- Build for Reuse
Continuous Integration (CI)

- Merge all work into shared mainline automatically
- Typically includes automated unit tests
- Develop, Stage, and Production environments
- Continuous Deployment: automatically deploy to machine if tests pass

Tools:
https://www.code-maze.com/top-8-continuous-integration-tools/
Code Reviews

Someone besides the author look at every line of code before merging

Code reviews look for:
1. Logical errors
2. Efficiency, maintainability
3. Documentation
4. Test effectiveness
5. Conformance to style guidelines

See:
wikipedia.org/wiki/List_of_tools_for_code_reviews
atlassian.com/agile/code-reviews
Documenting throughout

- Documentation = investment
  - Can reduce support costs
- Comments in Code
- User Guide and Developer’s Guide
- Wiki’s (confluence, etc.)
Tools

Static Analysis: Build it into your workflow


Linters: https://github.com/showcases/clean-code-linters

Profiling Tools

GUI Builders

Unit Testing Frameworks
Metrics

Logical Lines of Code & Lines of Comment (use calculator)
http://cloc.sourceforge.net

Number of files

Code review comments

Open issues

Requirement changes
Software Testing

HOW TO SPOT A TESTER IN A SUPERMARKET

HOW CAN I HELP YOU?
I'M GETTING AN ERROR AT THE SELF-SERVICE COUNTER
OH, DO YOU KNOW WHAT MIGHT HAVE CAUSED THE ERROR?
WELL, YES, IN THE PROMPT TO ENTER THE NUMBER OF BAGS I USED...

CUSTOMER SERVICE

...I ENTERED 4,294,967,297
CANS I ASK WHY YOU ENTERED THAT NUMBER?
I DON'T... KNOW

CUSTOMER SERVICE
Verification & Validation

Verification: Did you build it right (Requirements)

Validation: Did you build the right thing (Conops)
Testing Strategies

- Integration
- Unit
- Loading
- Stress
- Configuration
- Usability
- Installation
- Performance
- Security
- Recovery
- ...

...
Software Testing

- Make a plan at the beginning
- Test throughout
- Document throughout
- Review tests
Overall Recommendations

- Think about process—scale to project size, complexity, and organizational requirements
- Involve stakeholders, get buy-in
- Use existing design patterns, frameworks, and solutions
- Test throughout
- Automate when possible
- Code reviews save time!
- Don’t forget maintenance when planning
Resources

- NASA Software Engineering Handbook
- NASA Systems Engineering Handbook
- Software Engineering Body of Knowledge (SWEBoK)
- Systems Engineer Body of Knowledge (SEBoK)
- Project Managers Body of Knowledge (PMBoK)
- *Prognostics: The Science of Prediction*
- *The Pragmatic Programmer*
- IEEE
- Software Engineering Code of Ethics
Thank you

Dr. Kai Goebel (Discovery and Systems Health Lead, NASA Ames Research Center)

Dr. Matthew Daigle (Principal Data Scientist, NIO)

Dr. Abhinav Saxena (Senior Machine Learning Researcher, General Electric)

Dr. José Celaya (Lead Technologist for Data Science, Schlumberger)

Contact Chris Teubert (christopher.a.teubert@nasa.gov) with any questions