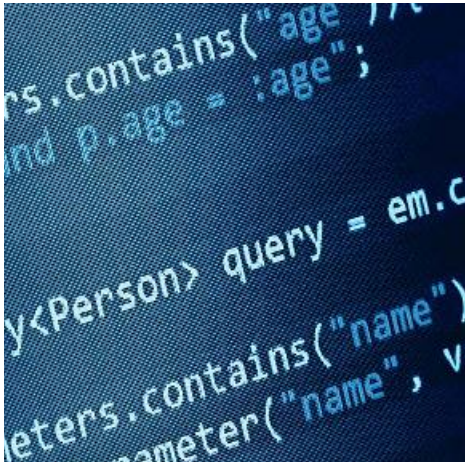
CS3500 Software Engineering

2017/2018



Dept. Computer Science
Dr. Klaas-Jan Stol

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Welcome to CS3500

Task 4

Haiku – format and examples

- 3 line poem format
- 5 – 7 – 5 syllables.

Build server is slow
Disk whirs, lights flash but no builds
Request SSD

Flakey code base
Instigate peer code reviews
No one has the time

Actions for Graded Task 4

Please work with your current team.

Graded Task 4:

Write an Architectural Haiku – a short document that follows the structure as outlined in the short article with the same title. The architecture to describe will be provided.

Goal:

To learn to focus on the most essential elements of an architecture, and to document those.

Architecture Haiku

Editor: **Eoin Woods**
Endava
eoin.woods@artechra.com

Architecture Haiku
A Case Study in Lean Documentation

Michael Keeling

SOFTWARE ARCHITECTS ARE taught to create comprehensive, complete documentation. We're taught to stamp out all assumptions and explicitly articulate our designs. Without restraint, however, architecture descriptions can become long-winded treatises on a system's design. In the course of being thorough, we often inadvertently obfuscate the architectural vision in documents that consequently aren't read.

An architecture haiku is a "quick-to-build, uber-terse design description" that lets you distill a software-intensive system's architecture to a single piece of paper.[1] Creating an architecture haiku is more than just an exercise in terse documentation. To write an effective haiku, you must focus on only essential design decisions and rationale. Much like its poetic cousin, an architecture haiku follows strict conventions and can express a mountain of information with a tiny footprint.

What Is an Architecture Haiku?

An architecture haiku aims to capture the architecture's most important details on a single piece of paper. Placing extreme constraints on the architecture description forces architects to focus on the design's most important aspects. With only one page to work with, there simply isn't space to waste on extraneous details.

For an architecture haiku to be comprehensive, Fairbanks recommended that it include[1,2]

- a brief summary of the overall solution,
- a list of important technical constraints,
- a high-level summary of key functional requirements,
- a prioritized list of quality attributes,
- a brief explanation of design decisions, including a rationale and tradeoffs,
- a list of architectural styles and patterns used, and
- only those diagrams that add meaning beyond the information already on the page.

An important key to success is to focus on the most essential ideas for each item on this list. If something isn't important, don't include it. If an idea requires more detail, don't skimp—but don't overdo it. The point isn't to omit or reduce critical information but to highlight it. Removing noise that might otherwise drown out the most important ideas helps shine a spotlight on them.

Architecture haiku creates a *cognitive trigger* that facilitates the recall of essential contextual information about a design decision. Rather than fully describing a decision using comprehensive prose and diagrams, assume the reader already has much of the required knowledge, and focus on the critical details.[2]

Two of the most common examples of a cognitive trigger are architectural styles and patterns. With a single word

0740-7459/15/$31.00 © 2015 IEEE

Actions for Graded Task 4

Input:

- The IEEE Software article that describes what an Architecture Haiku is.
- A paper that presents an architecture.
  - To be provided!

Process:

- Read the IEEE Software article, and get familiar with the format of an architectural haiku.
- Write the Architecture Haiku ☺

Contents of the haiku (see paper)

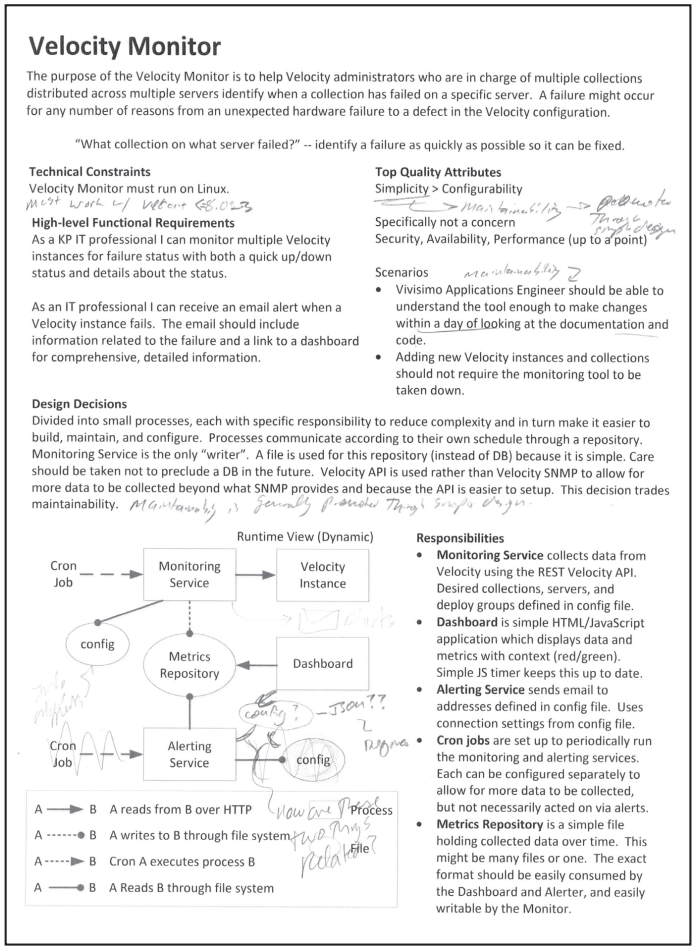
- a brief summary of the overall solution
- a list of important technical constraints
- a high-level summary of key functional requirements
- a prioritized list of quality attributes
- a brief explanation of design decisions, including a rationale and tradeoffs
- a list of architectural styles and patterns used
- only those diagrams that add meaning beyond the information already on the page.

Actions for Graded Task 4

Output:

An architectural Haiku.

(see next 2 slides for bigger version + see IEEE Software paper that contains the original)



Velocity Monitor

The purpose of the Velocity Monitor is to help Velocity administrators who are in charge of multiple collections distributed across multiple servers identify when a collection has failed on a specific server. A failure might occur for any number of reasons from an unexpected hardware failure to a defect in the Velocity configuration.

"What collection on what server failed?" -- identify a failure as quickly as possible so it can be fixed.

Technical Constraints
Velocity Monitor must run on Linux.

High-level Functional Requirements
As a KP IT professional I can monitor multiple Velocity instances for failure status with both a quick up/down status and details about the status.

As an IT professional I can receive an email alert when a Velocity instance fails. The email should include information related to the failure and a link to a dashboard for comprehensive, detailed information.

Top Quality Attributes
Simplicity > Configurability

Specifically not a concern
Security, Availability, Performance (up to a point)

Scenarios

- Vivisimo Applications Engineer should be able to understand the tool enough to make changes within a day of looking at the documentation and code.
- Adding new Velocity instances and collections should not require the monitoring tool to be taken down.

Design Decisions
Divided into small processes, each with specific responsibility to reduce complexity and in turn make it easier to build, maintain, and configure. Processes communicate according to their own schedule through a repository. Monitoring Service is the only "writer". A file is used for this repository (instead of DB) because it is simple. Care should be taken not to preclude a DB in the future. Velocity API is used rather than Velocity SNMP to allow for more data to be collected beyond what SNMP provides and because the API is easier to setup. This decision trades maintainability.

Runtime View (Dynamic)

Responsibilities

- **Monitoring Service** collects data from Velocity using the REST Velocity API. Desired collections, servers, and deploy groups defined in config file.
- **Dashboard** is simple HTML/JavaScript application which displays data and metrics with context (red/green). Simple JS timer keeps this up to date.
- **Alerting Service** sends email to addresses defined in config file. Uses connection settings from config file.
- **Cron jobs** are set up to periodically run the monitoring and alerting services. Each can be configured separately to allow for more data to be collected, but not necessarily acted on via alerts.
- **Metrics Repository** is a simple file holding collected data over time. This might be many files or one. The exact format should be easily consumed by the Dashboard and Alerter, and easily writable by the Monitor.

Velocity Monitor

The purpose of the Velocity Monitor is to help Velocity administrators who are in charge of multiple collections distributed across multiple servers identify when a collection has failed on a specific server. A failure might occur for any number of reasons from an unexpected hardware failure to a defect in the Velocity configuration.

"What collection on what server failed?" -- identify a failure as quickly as possible so it can be fixed.

Technical Constraints

Velocity Monitor must run on Linux.

Must work w/ Vel 1.8.0_73

High-level Functional Requirements

As a KP IT professional I can monitor multiple Velocity instances for failure status with both a quick up/down status and details about the status.

As an IT professional I can receive an email alert when a Velocity instance fails. The email should include information related to the failure and a link to a dashboard for comprehensive, detailed information.

Top Quality Attributes

Simplicity > Configurability

→ Maintainability → Automated Tests, simple design

Specifically not a concern

Security, Availability, Performance (up to a point)

Scenarios maintainability 2

- Vivisimo Applications Engineer should be able to understand the tool enough to make changes within a day of looking at the documentation and code.
- Adding new Velocity instances and collections should not require the monitoring tool to be taken down.

Design Decisions

Divided into small processes, each with specific responsibility to reduce complexity and in turn make it easier to build, maintain, and configure. Processes communicate according to their own schedule through a repository. Monitoring Service is the only "writer". A file is used for this repository (instead of DB) because it is simple. Care should be taken not to preclude a DB in the future. Velocity API is used rather than Velocity SNMP to allow for more data to be collected beyond what SNMP provides and because the API is easier to setup. This decision trades maintainability. Maintainability is generally promoted through simple design.

Runtime View (Dynamic)

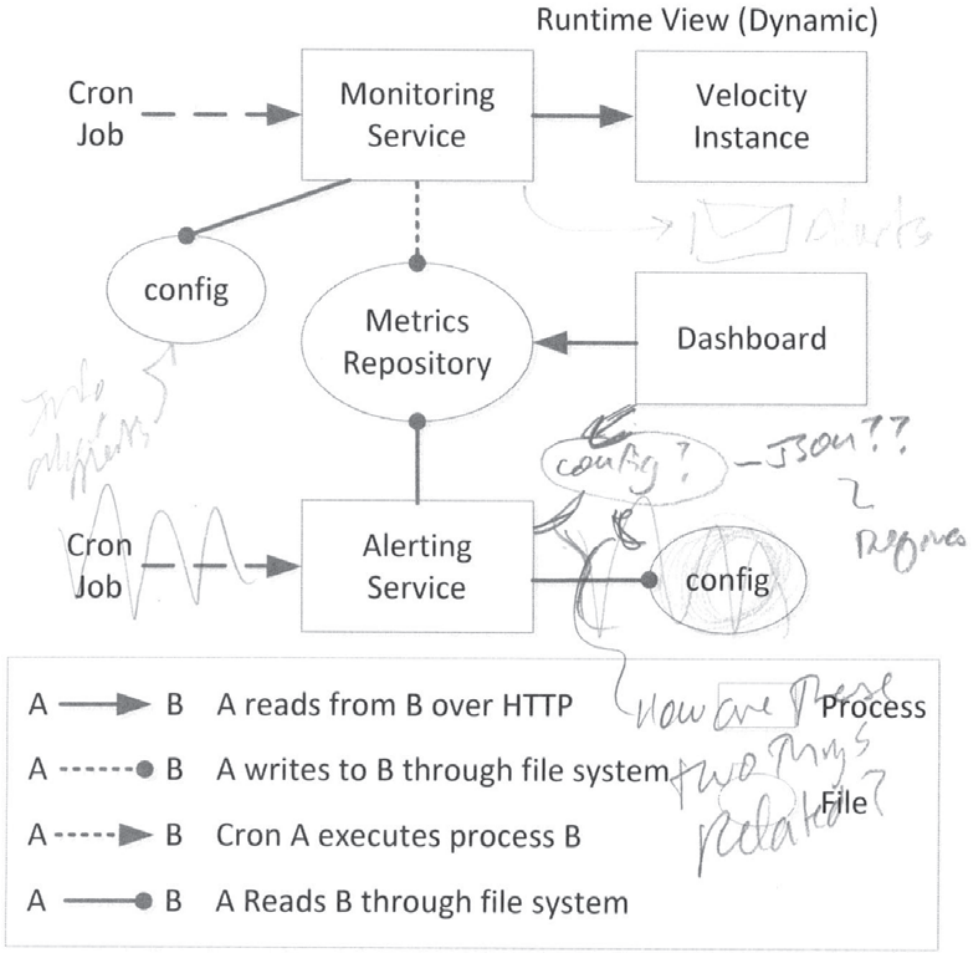
Responsibilities

- **Monitoring Service** collects data from Velocity using the REST Velocity API

should not require the monitoring tool to be taken down.

Design Decisions

Divided into small processes, each with specific responsibility to reduce complexity and in turn make it easier to build, maintain, and configure. Processes communicate according to their own schedule through a repository. Monitoring Service is the only "writer". A file is used for this repository (instead of DB) because it is simple. Care should be taken not to preclude a DB in the future. Velocity API is used rather than Velocity SNMP to allow for more data to be collected beyond what SNMP provides and because the API is easier to setup. This decision trades maintainability.



Responsibilities

- **Monitoring Service** collects data from Velocity using the REST Velocity API. Desired collections, servers, and deploy groups defined in config file.
- **Dashboard** is simple HTML/JavaScript application which displays data and metrics with context (red/green). Simple JS timer keeps this up to date.
- **Alerting Service** sends email to addresses defined in config file. Uses connection settings from config file.
- **Cron jobs** are set up to periodically run the monitoring and alerting services. Each can be configured separately to allow for more data to be collected, but not necessarily acted on via alerts.
- **Metrics Repository** is a simple file holding collected data over time. This might be many files or one. The exact format should be easily consumed by the Dashboard and Alerter, and easily writable by the Monitor.

Graded Task : Contents of Deliverable

(Similar to deliverable D1/D2/D3)

- Cover page with team ID + team member names + student ID numbers.
- The architectural Haiku
- On a separate page: describe contributions of team members – format as you see fit.
- Think of formatting
  - make it look OK – don't use Notepad

Graded Task 4

- Follow the formatting instructions in the IEEE Software article – e.g. fonts not smaller than 10 pt.
- It must be readable!
- Submission must be in PDF format.
- Submit through Moodle by November 23. (2 weeks from today – Nov. 9)
- Please: Only 1 submission per team.

Questions?

Please email: k.stol@ucc.ie

Thank you for your attention

Questions & suggestions can be sent to:

k.stol@ucc.ie