The Travails of an OO Design

Gregory Hartman
Lessons Learned

• Good designs are valuable
• Industrial SE practices are primitive
• Performance and design are compatible
• Profilers are limited
• Refactoring a good design is cheap
• The STL creates performance problems
Prior Experience

- Unisys: 2.5 years
- HDS: 2 years
- Connectware: 1.5 years
- ??: 4 years

~ 10 years
Management Worldview

- Minimize programmer salaries
- Assign programmers to code permanently
- Priorities are schedule, features, quality
- Design is a waste of time
- No need for metrics
- Search for magic bullet technology
Resulting Software

• Called an OO design, but...
• Little inheritance
• Uneven decomposition
  – Functions with >1000 LOC
  – 8 levels of nested control structures
  – Data format details not encapsulated
  – Myth: decomposition leads to slow code
• Little design documentation
Consequences

• The code was difficult to modify
  – A data format change took 6 months
  – Still working out the bugs one year later

• Difficult to reuse code

• Programmers burnt out

• Requests for enhancements turned away
Project Goals

• Reduce latency of maps over internet
• Convince management to support design
• Debunk performance myth
• Create a maintainable code-base
Project Approach

- Reengineer the system
- Use UML techniques
- Drive design with carto. requirements
- Team of 2 core designers (including myself)
- Consulted with about 5 domain experts
- Implemented in C++ with reference counting
Research Objectives

• Test case for applying UML
• Evaluate the performance of OO designs
  – Remove bottlenecks
• Compare latency of server-side and client-side mapping systems
Features: Polygons
Features: Lines
Features: Points
Label Placement
Reuse and Scaling
Feature Reduction
Multiple Layers
Databases

- **SpatialDatabase** – Holds permanent data
  - Name, shape, etc.
  - Indexed by (feature type, geographic area)
  - Must clip large objects to a reasonable size

- **StyleDatabase** – Formatting information
  - Colors, widths, etc.
  - Draw order
  - When to drop features
  - Label Placement priority
Layers

• Combines Data from SpatialDatabase and StyleDatabase to form a complete layer
Map

- Combines multiple layers
- Provides simple interface to programmers
LabelPlacement

- Intercept and buffer points
- Draw other features immediately
- Store feature locations
- Calculate acceptable label positions
- Place highest priority first

Diagram:

MapLayer ---> Map ---> LabelPlacement ---> Graphics
Extensibility

• Problem: Both databases will be extended with new fields
  – Classes that don’t understand just pass through
• Solution: a Feature class to hold the data
  – Examples: points, lines, and polygons with formatting
  – Has name for the label
  – Some number of points
• Feature objects are passed between classes
Efficiency

- Could have about 100,000 features/map
  - Why do all of this allocation and de-allocation
- Grouping helpful in other parts of the code
  - Graphics API’s want groups
- SpatialDatabases often have them
- Created a class called FeatureGroup
  - Interface mimics a container of features
  - Also presents grouping interface
Standard Configuration

Map

MapLayer ➔ LabelPlacement ➔ Graphics

StyleDatabase

SpatialDatabase
Research Configuration

- Map
  - MapLayer
    - StyleDatabase
  - LabelPlacement
  - Graphics
    - HttpClient
    - HttpServer
    - SpatialDatabase
Initial Results

• Forced to implement with partial design
• Created fair documentation
  – Use cases, class diagrams, sequence diagrams
  – Not updated during implementation
  – Lacked reasons for some crucial decisions
• Effort: 3 months design, 3 months coding
  – Management did not value design effort
• Design is iterative
Perceived Performance

- Compared well with previous system:
  - Half the previous draw time in general case
  - Much faster on detailed maps (2s vs. 30s)
  - New draw times were 0.5s-2s
- Management happy, but …
Performance Issues

• Indications of inefficiency
  – Drawing or LabelPlacement should take time, but didn’t
  – Execution time spread throughout code instead

• Profiler (Quantify) didn’t show hot-spots
  – Stack too deep
  – Inline functions hide the code
My Next Project

• Evaluation of Curl
• What is Curl?
  – A Java-like programming language out of MIT
  – Optimized for downloading programs to web browsers
  – No byte-code
  – Code compiled as it is downloaded
  – For more information, see www.curl.com
Approach to the Evaluation

• Chose our project as a test case for Curl evaluation
• Decided to remove the interfaces
  – Performance of interfaces suspect anyway…
• Design didn’t matter
  – Code would never be used
  – Tight timeframes
Initial Curl-Based System

• Ran in about 2-3s
  – Well, it’s not C after all…
• The vendor (Curl) wasn’t happy
  – So I took a business trip
• Led to the vendor
  – Choosing better API functions for drawing
  – Optimizing their APIs and compiler
• Result: draw time between 0.1s-0.2s
  – Zero perceived response time
Revisiting the Design

• 10x performance from Curl-based system
  – Zero response time compelling
• Dropped pieces of code to find the culprit
  – Aside: our good design made this easy
  – Improvement when LabelPlacement dropped
    • LabelPlacement used a container
• Optimized STL container usage
• Combined LabelPlacement and MapLayer
  – This breaks the design
Reflection

• Things I wish I had known:
  – Market forces / motivations of companies
  – Not to accept information at face value
  – How to balance SE vs. business tradeoffs
  – How to design for performance

• Did we drop the right parts of the design?
  – Software Engineering triage?
  – When do you just need to walk away?
Promoting Software Engineering

• What technical metrics do we need?
  – Are design-time metrics feasible?
• How do we express this to management?
  – Bridging technical metrics to the bottom line
• How do we evaluate design tradeoffs?
  – Good-design vs. good-performance tensions
  – Good-design vs. time-to-market tensions
• Need elastic (give & take) SE techniques
Goal: Understand Containers

• Is the container problem general?
  – Stack problems may have been a STL artifact
  – Inline functions make it hard to total time
  – Would automatic garbage collection be faster
  – Can STL provide useful hints to the profiler

• How do we create tools to find out?
  – Implementing a new system seems expensive
  – An accounting perspective: % of useful work?
  – The problem is API abuse looks useful
Conclusion

• Email:
  – gghartma@cs.cmu.edu

• Presentation