

An Analysis of LifeV

Christophe Prud'homme

École Polytechnique Fédérale de Lausanne

Milan, January 31 2004

Co-Authors

- D. A. Di Pietro
- G. Fourestey
- C. Winkelmann

Analysis Steps

Characteristics

- Internal : Strengths, Weaknesses
- External : Opportunities, Threats

Technical characteristics of a good code

- maintainability
- extendibility
- efficiency
- robustness
- capitalization : maintainability + extendibility
- cheap development cost

*It is not that hard to develop a code,
it is way more difficult to maintain it.*

Questions

- What advantages do we have?
- What do we do well?
- What relevant resources do we have access to?
- What do other people see as our strengths?

LifeV Strengths

- Applied mathematicians community
- High number of developers involved
- State of the art algorithms
 - Algebraic factorisation for Stokes
 - FSI
 - IP stabilization
 - 3D fem level set implementation
- Access to intensive scientific computing platforms
- Very powerful open programming environment
- Many ways to get information about the project

Questions

Questions

- What could we improve?
- What do we do badly?
- What should we avoid?

It is best to be realistic now, and face any unpleasant truths as soon as possible.

LifeV Weaknesses

Weaknesses

- Often little background in computer science and developing/maintaining a library is a very challenging task.
- Three sites: Paris, Milan, Lausanne
- Lack of communication, coherence and consistency
- Lack of common objectives, common rules and common spirit
- More and more people : lack of coherence

LifeV Weaknesses

- Use too many libraries (but only two are mandatory), LifeV is too complicated to build
- Some people have difficulties with programming environments : compilers, libraries, tools. . .
- LifeV is often *under-engineered*
- List of issues : Design Debt
 - Spaghetti code
 - Large portion of unnecessarily complicated code
 - Lack of encapsulation
 - Lack of factorization, code duplication
 - Bad coding practices



Questions

Questions

- Where are the good opportunities facing you?
- What are the interesting trends you are aware of?

LifeV Opportunities

Opportunities

- EU Project Haemodel project to initiate/stimulate/finance development
- Other related EU Projects : a way to capitalize on the development
- Industrial contracts : MOX, INRIA, EPFL
- Relations with local hospital (CHUV) to get medical data and feedback
- Will maybe used in the Alinghi project for free surface flows
- Could be used for teaching purposes
- Attract more people/developers/students, thanks to talks and <http://www.lifev.org>

Questions

- What obstacles do we face?
- What is our competition doing?
- Is changing technology threatening our position?
- Could any of our weaknesses seriously threaten our work?

LifeV Threats

- Diverging interests could threaten the achievements of the objectives
- Many opensource FEM code that have better design, more physics. . .
- Mathematicians are usually not computer scientists
- Repel people and students because the code does not provide a good programming experience

Some Propositions

Propositions

- Define objectives, audience, spirit
- Increase communications towards senior developers
- Layers of access in the code
 - Developers are initially granted with a low write access level
 - If they must access read-only parts, they send a patch
 - They get to higher levels after showing satisfactory code
 - Issues: feedback speed, loss of interest
- People should feel responsible of what they do (e.g rules and spirit)
- C++ and computer science courses

Some Propositions

Some Propositions

- Constant *refactoring* to keep design debt low ▷
- Use TDD: Test-Driven Development and Continuous refactoring ▷
- Stable but good interfaces

END

Thoughts? Comments? Propositions?

Under-Engineering: Causes



Causes

- We don't have time, don't make time or aren't given time to refactor
- We aren't knowledgeable about good software design
- We are expected to quickly add new features to existing systems
- We are made to work on too many projects at once

Under-Engineering: Symptoms



Symptoms

- Everything talks to everything
- Data may be global
- Data may be passed by back channels to circumvent the system's original structure
- Variable and function name might be uninformative or misleading
- Code is duplicated
- Functions are lengthy, convoluted and perform unrelated tasks
- Code unreadable or even undecipherable

Under-Engineering: Consequences



Consequences

- increase the design debt
- Expensive
- Difficult to maintain , unmaintainable
- Leads to “fast, slow, slower” rhythm
 - Deliver 1.0 with junky code
 - Deliver 2.0 but junky code slows you down
 - Future releases, you go slower and slower as junky code multiplies until people lose faith in the system
 - Somewhere around 4.0, you realize you can't win and you consider a total rewrite

Design Debt



How?

You don't consistently do 3 things:

- Remove duplication
- Simplify your code
- Clarify your code's intent

When?

- Ignorance or commitment to “don't fix what ain't broken”
- Design debt incur late fees
- If you don't pay late fees, you pay higher late fees
- The more you don't pay, the worse your fees and payments become
- Eventually getting out of debt becomes an impossible dream

Refactoring

Motivations

- make it easier and cheaper to add new code
- improve the design of existing code
- gain better understanding of code
- make coding less annoying



Test Driven Development



Process

- Create a test that expresses what you expect the code to do
- Program whatever is expedient to make the test pass
- Refactor : Improve the design of the code that passes the test

Features

- Lean iterative and disciplined style of programming
- Maximize focus, relaxation and productivity
- Keep defect counts low
- Refactor without fear
- Produce simpler and better code