Can software be healthy?

Jesus M. Gonzalez-Barahona

Universidad Rey Juan Carlos
@jgbarah http://jgbarah.github.io/presentations

SoHeal 2019
Montreal (Canada), May 28th 2019

Health,
what is health?
Can anyone be healthy
at all?

Speaker: What do we want?
Crowd: Patience!
Speaker: When do we want it?
Crowd: Right now!!!

Adapted from a well known joke
by Eugenio (Spanish humorist).

The theory

Software should behave
according to requirements,
be cheap to maintain,
be easy to use,
have good performance,
...

"We want software of good quality"

The practice

In most cases...

- Functionality: shallow verification
- Requirements: from nonexistent to incomplete
- Maintainability: very expensive
- Usability: many facets
- Performance: only a relative target

"Good enough", depending on the stakeholder

Improving quality

The quest for quality

"Traditional" approach in software engineering:

- Product quality
  (ISO 9126, CISQ)
- Process quality
  (ISO 9001, CMM)

Follow the rules, increase quality

CISQ (code) quality model

- reliability
- efficiency
- security
- maintainability

https://www.it-cisq.org

CISQ (code) quality model

SOFTWARE QUALITY CHARACTERISTIC	CODING PRACTICES UNIT LEVEL	ARCHITECTURAL PRACTICES SYSTEM LEVEL
RELIABILITY	• Protecting state in multi-threaded environments • Safe use of inheritance and polymorphism • Resource bounds management, Complex code • Managing allocated resources, Timeouts	• Multi-layer design compliance • Software manages data integrity and consistency • Exception handling through transactions • Class architecture compliance
PERFORMANCE EFFICIENCY	• Compliance with Object-Oriented best practices • Compliance with SQL best practices • Expensive computations in loops • Static connections versus connection pools • Compliance with garbage collection best practices	• Appropriate interactions with expensive or remote resources • Data access performance and data management • Memory, network and disk space management • Centralized handling of client requests • Use of middle tier

Measuring quality

There are other motivations

What if the focus is "knowing" instead of "improving"

- comparison
- tracking
- self-awareness

There are other subjects

What if the people are also important?

- the builders
- the evaluators

The builders

Specially important in FOSS:

- diverse people working together
- different motivations, agendas...
- the sense of community

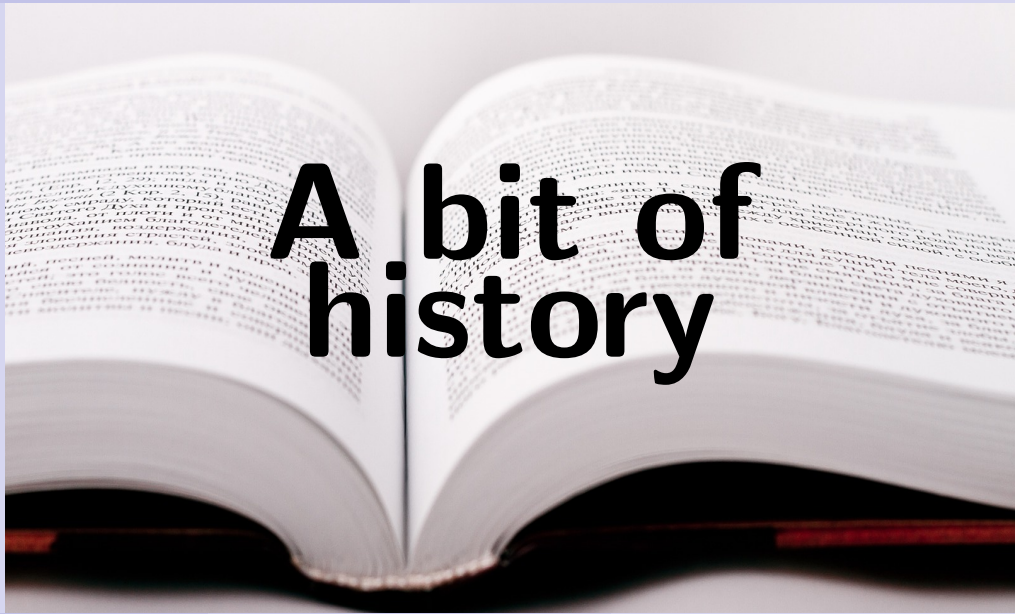
The evaluators

Different goals / interests
mean
different definitions of "good"

And we still have the context...

Software is not used in a vacuum:

- legalese
- support
- economy
- ecosystem
- ...



A bit of history

OpenBRR

Business Readiness Rating™

A Framework for Evaluating Open Source Software

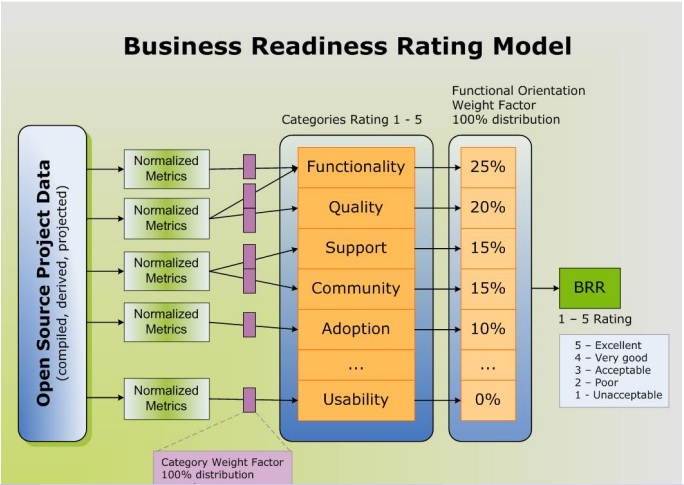
STEERING COMMITTEE

Larry Augustin, Open Source Strategist
Michael Goulde, Forrester Research
Peter Kronowitt, Intel
Murugan Pal, SpikeSource
Josh Berkus, PostgreSQL
Marc Hedlund, O'Reilly CodeZoo
George Pace, Prudential Financial
Anthony Wasserman, Carnegie Mellon West (Chair)

FOUNDING SPONSORS

Carnegie Mellon West
spike SOURCE
intel.
O'REILLY code Zoo

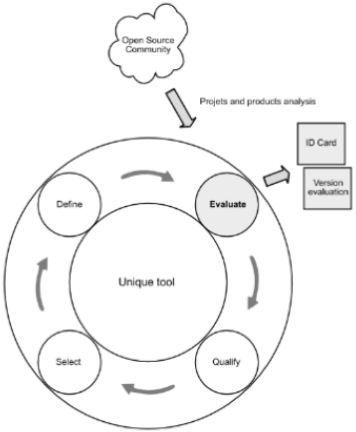
OpenBRR



Atos Origin

QSOS

QSOS



- ID card and version evaluations;
- Scoring of criteria on three major axis:-
  - Functional coverage;
  - Risks from customer perspective;
  - Risks from Atos Origin perspective;
- Weighted metrics for product scoring;

QSOS

Intrisic robustness
- Maturity
- Adoption
- Development Roadmap
- Activity
- Development independence

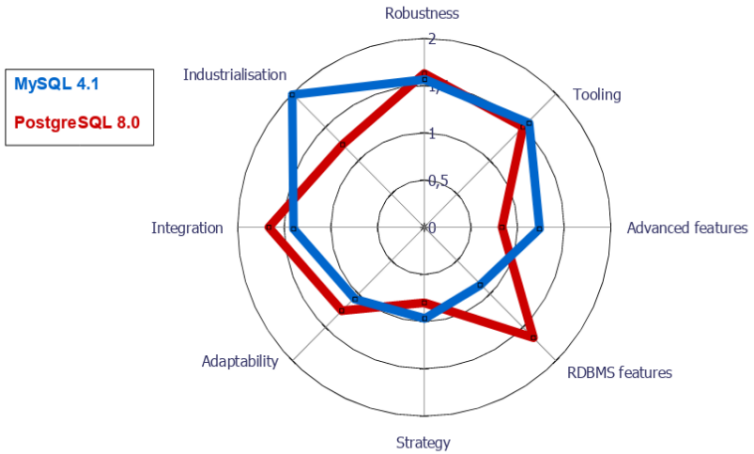
Industrialised solution
- Services
- Documentation
- Quality Assurance
- Exploitability

Integration
- Adherence to standards
- Interface with other products

Strategy
- Licence
- Copyright owners
- Modification of source code
- Roadmap
- Sponsor

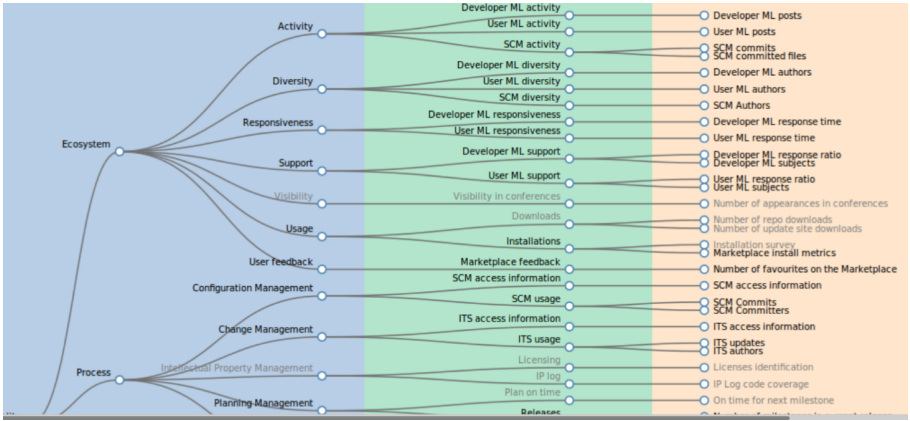
Technical adaptability
- Modularity

QSOS



Polarsys Quality Model

Polarsys Quality Model



Software health

"A set of **characteristics**
of a software **project**
determining its capability for producing
software of good quality,
according to certain **criteria**

What is software health?

A concept applied to a **project**:

- Criteria to define quality
- Characteristics that allow for that quality
- Time spot for measuring

Measuring software health

- Quantify quality criteria
- Find indicators that summarize criteria
- Find values for them that characterize health
- Track their evolution

Example

- Criteria for quality: minimize unfixed errors
- Indicator: unfixed bug reports
- Healthy value: X unfixed bug reports per KLoC
- Alarm when number below X

The causes for health

The really interesting matter
is to know the causes
for variation in indicators

Example: unfixed bug reports are minimized by good code review

Some ideas

On the shoulders of giants

Systems are composed of many modules:

- Dependencies matter
- Overall health dependent on health of all components
- In some cases, dependent on health of the most unhealthy component
- Projects and communities are interdependent

Making decisions for tomorrow

Many systems are in production for many years:

- Prediction on future health
- Not all aspects are equally relevant (example: fixing bugs vs. new functionality)
- Important: understanding dynamics (extending past to future is not good enough)

Integrating metrics with development

Can health be yet another factor to consider?

- It could be an indicator for every stakeholder
- Computed frequently, so that it is up to date
- Published widely, so that everyone is aware

Include health in the data for decision making

Working with stakeholders

- Builders
- Integrators
- Users

Health for different actors
for different purposes

http://chaoss.community

Understanding dynamics

How do specific actions impact
on the health model
for a software development system?

Towards a new research framework

Define health conditions
Find out how to measure indicators of health
Study deviations from healthy conditions
Learn how to help to go back to healthy
Include all of this in the development process

Simple example

Health condition: no regressions
Indicators: tests failing
Deviations: old errors appear
Mitigation: automatic testing

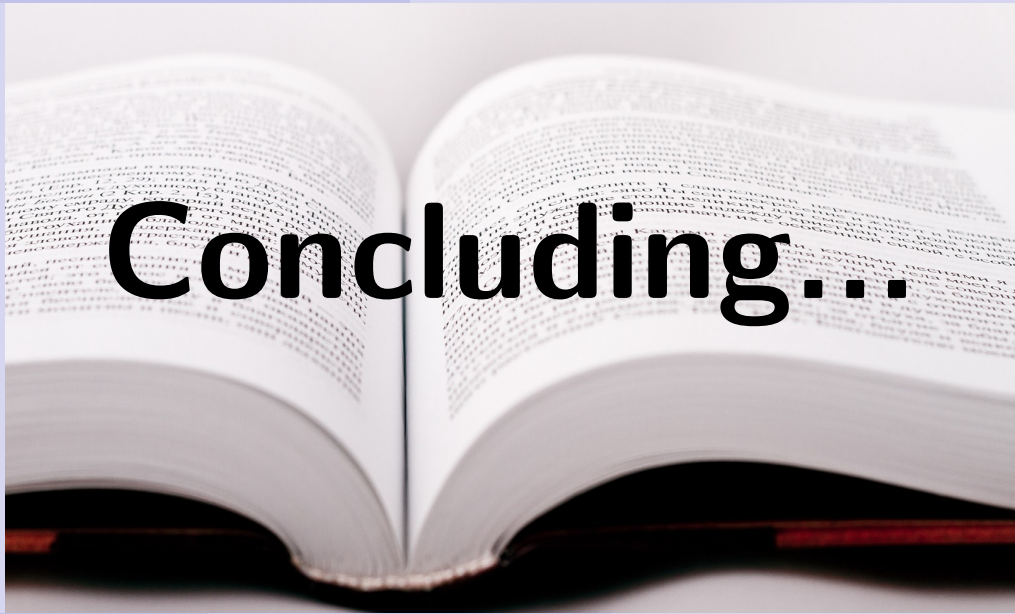
Continuous integration system

Beyond opinions

Evidence that the indicator shows deviation from healthy condition

Evidence of mitigation:

- condition go back to healthy
- indicator go back to normal



Concluding...

Can we do this
in non-trivial cases?

Software health may provide
a good framework
for structuring research,
producing useful analysis,
and producing actionable outputs

This document (including source) is available from
https://jgbarah.github.io/presentaciones