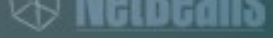
Designing API:
20 API Paradoxes

Jaroslav Tulach
NetBeans Platform Architect



Motto

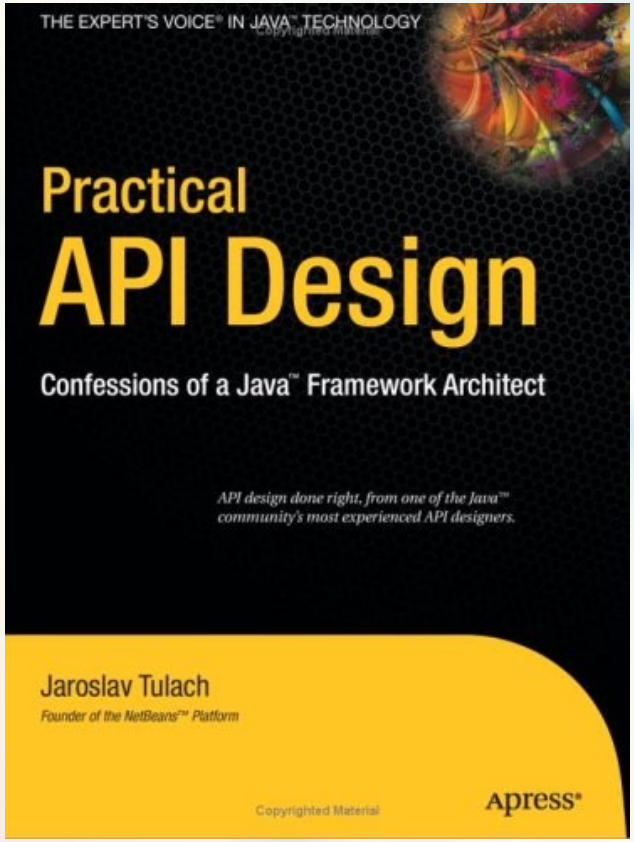
Just like there is a difference between describing a *house* and describing a *Universe*, there is a difference between writing a *code* and producing an *API*.

About Me

- 1996 – Xelfi @ MatFyz
- 1997 – Initial NetBeans APIs
- 1999 – Acquired by Sun Microsystems
  - 2008 - Practical API Design book
- 2010 – Acquired by Oracle
  - NetBeans & JDeveloper
  - 2012 – 20 API Paradoxes book
- now – Java & Browser

The First Book

“European” version



http://apidesign.org

Paradox

Is paradox unnatural?

- Crossing the knowledge horizon
  - Fear of unknown
  - I know it “all” mode
- Expectation vs. Reality
  - The less “fear” the more paradoxes
- Software knowledge
  - School
  - In-house development
  - Framework

More Organized Book

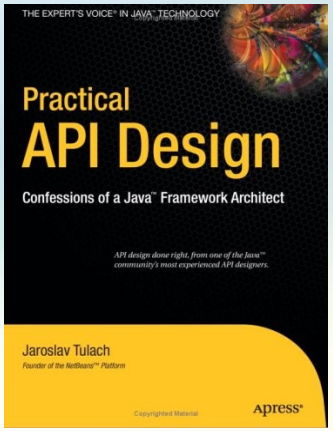
“U.S.” version

http://apidesign.org

20 Paradoxes to Talk About

Seek for More

Q&A

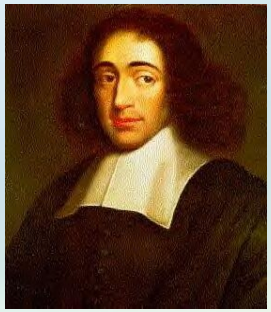


http://www.apidesign.org/

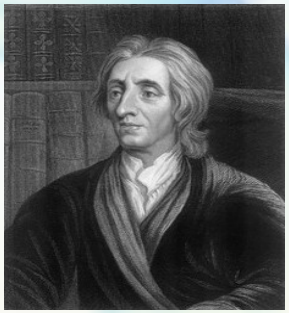
Who Are Your Users?

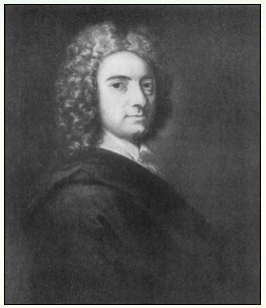
Rationalists?





Empiricists!?





Clueless!



Selective Cluelessness

One cannot understand everything

- Understanding is limited
  - takes time
  - brain is finite
- Not necessary to understand everything
  - Linux, Apache, MySQL on the stack
  - Learn just the API facade
- Minimize Understanding!
- Make it increasable!

http://wiki.apidesign.org/wiki/Cluelessness

What is API?

Just like writing a book

- One writer
  - Design in committee?
- Many readers
  - Unknown to the writer
  - Envisioned via use-cases
- Best-seller
  - Speak clearly
  - Built up on reader's knowledge
  - Keep consistency

http://wiki.apidesign.org/wiki/APITypes

Maintaining an API

Develop and sustain!

- Write once and publish
  - Creativity is good
  - Strive for elegance
- Switch to sustaining mode
  - Preserve made (unknown) investments
  - Polish
  - Promote
- Incremental API Design
  - Get ready for evolution

http://wiki.apidesign.org/wiki/Evolution

Quality of an API?

3 sides to every API

- Writer's point of view
  - > Sacrifice
  - > Elegance is the least priority
- Users' point of view
  - > API usage shall lead to “nice” code
  - > Upgrade breaks no existing code
- Essential API “goodness”
  - > Correctness (via usecases)
  - > Stability (via tests)
  - > Isolate writer and reader

http://wiki.apidesign.org/wiki/3SidesToEveryAPI

Good Technology

Holy Grail every vendor seeks

- Coolness
  - Attracts attention
  - Otherwise useless
- Time to Market
  - Achieve more by doing/knowing less
  - Cluelessness
- Cost of Ownership
  - Evolution
  - Compatibility

http://wiki.apidesign.org/wiki/Good_Technology

Time Matters

Compatibility with previous releases

- Source compatibility
  - JavaScript, PHP – no binaries
  - Knowing the language is enough
- Binary compatibility
  - JAR, object files, assemblies
  - Understand the ABI rules
- Functional compatibility
  - Tests, tests, tests
- The invisible job

http://wiki.apidesign.org/wiki/BackwardCompatibility

Source compatibility

What compiled needs to compile

- Source compatibility gotchas
  - Making **protected** method **public**
  - Adding overloaded methods
  - Wildcard imports collisions
- Beware of “patch” compatibility
  - Close proximity of MediaWiki

http://wiki.apidesign.org/wiki/BackwardCompatibility

Binary compatibility

What linked together needs to link

- Most important type for Java, C, etc.
  - Compile with oldest vs. run with newest
- Some paradoxes
  - Making **protected** method **public** is OK
  - Adding overloaded methods is OK
  - Wildcard imports collisions cannot happen
- Some gotchas
  - Changing type of field or method
  - Adding virtual method in C++
- Signature testing tools

http://wiki.apidesign.org/wiki/BackwardCompatibility

Functional compatibility

The ultimate goal is that the system shall work!

- Automated tests
  - Test coverage
  - Sample API usage
- Multi-threading
  - Never call foreign code with a lock
  - Beware of re-entrant calls
  - Emulate deadlocks in tests
- Memory management
  - Injection of references
  - Test for proper clean up with assertGC

http://openide.netbeans.org/tutorial/test-patterns.html

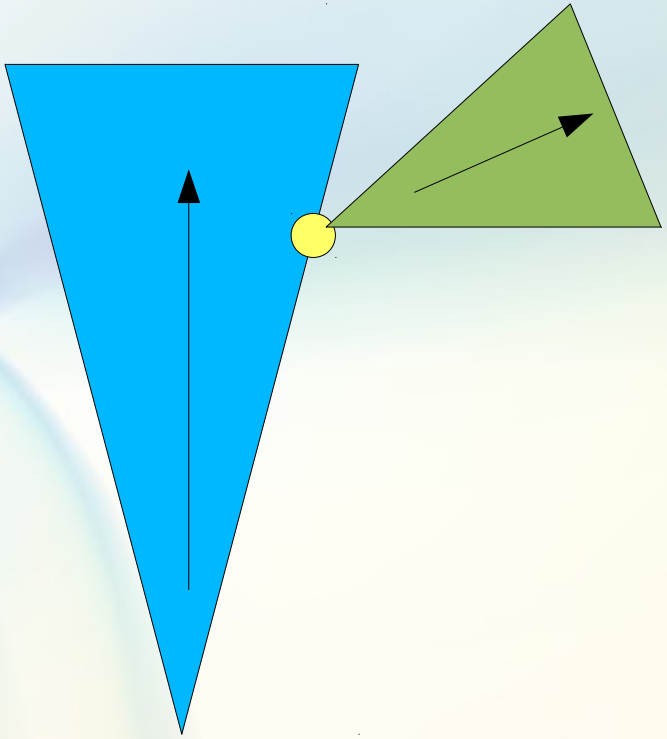
Factorial

Demo

Client vs. Provider

Evolution is different

- API for clients to call
  - > “Open space”
  - > Can grow with time
- API to implement
  - > Cannot change
  - > A “fixed point”
- Don't mix
- Compose
  - > PropertyChangeListener and Event



http://wiki.apidesign.org/wiki/ClientAPI

Code Against Interfaces

The Java misinterpretation

- Review API before publishing
- Recognize API from implementation
- Old advice
  - Interface means abstract definition
  - Not Java **interface** keyword
- Evolution aspects
  - Interfaces better for “fixed points”
  - (**final**) classes better for “open spaces”

http://wiki.apidesign.org/wiki/Chapter_6

Maintenance cost

How hard is to maintain an API?

- API happens
  - > Distributed teams need it
- No users => no bugs => no work
- Feature requests
  - > Let your users implement them
- Bugs
  - > Request automated test by reporters
- Maintaining an API is simpler than maintaining code with no API

http://wiki.apidesign.org/wiki/CodeInjection

API Review

Rejecting "ugly" API changes?

- Allow anyone propose API change
  - Public rules
- Checklist
  - Use-case driven
  - Enough test coverage
  - Properly documented
  - Backward compatible
- Give up on beauty
  - API design is not art!

http://wiki.apidesign.org/wiki/CodeInjection

Alternative Behavior

Balance bug fixes and compatibility

- Compile-time
  - New constructor, factory, setter
- Deploy-time
  - Per VM configuration
- Side by side
  - Copy the old class into new
  - Prevents mutual exchange
- Runtime-time
  - Inspect caller's expected environment

http://wiki.apidesign.org/wiki/AlternativeBehaviour

Modularity

Exactly specify code's environment

- Code does not live in vacuum
  - Needs appropriate environment
- Libraries evolve in time
  - Identify them with version number
- One can always mimic old environment
  - Alternative Behaviors
  - Emulation layers
  - Bridges

http://wiki.apidesign.org/wiki/Modularity

APIs Are Like Stars

Sent your old API to black hole!

- Can one get rid of old API?
  - > While keeping backward compatibility?
- Yes, due to modularity
  - > Release new library version
  - > Mimic old behavior until clients migrate
  - > All migrated => old behavior is gone
- Place for beauty
  - > Old, ugly API can compatibly disappear

http://wiki.apidesign.org/wiki/Star

Research Field

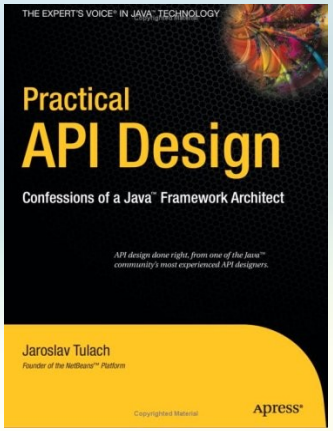
Place for Rationalistic Souls

- NP-Complete problems
  - 3SAT to Modular configurations
- Verification
  - Signature checks
  - Is an upgrade safe?
- Language Design
  - Modifiers are misleading
  - Distributed Modularity

http://wiki.apidesign.org/wiki/LibraryReExportIsNPComplete

Seek for More

Q&A



http://www.apidesign.org/