Love your architecture II

SATURN 2017

Paulo Merson
A not so uncommon story...
A software architecture was carefully defined based on

- architecture and design patterns
- experience with similar design problems
- discussions with several stakeholders
The architecture was incrementally refined and implemented.
Sprint after sprint, code became the main artifact.
Development and maintenance

• spanned several years
• involved tens of people
The end of the story...
Actual architecture != intended architecture
Lack of architecture conformance

• makes the code hard to understand and change, mainly because of undesired dependencies

• may affect portability, performance, interoperability, security and other qualities
How to avoid code and architecture disparity?
1) Communicate the architecture to developers
   • Not the focus of this presentation
2) Automate architecture conformance analysis
   • Often done with static analysis tools
Two limitations of static analysis tools, as commonly used…
1) Many people just enable built-in checks, which
   • are oblivious to your architecture design constraints

2) Developers can ignore reported violations
   • There’s no time to “clean the house”
   • There may be no accountability for adding violations
How did we overcome limitation 1?
We created *custom checks*
How did we overcome limitation 2?
Upon code commit a *pre-commit hook*
- runs custom checks on each source file

If a custom check detects a violation
- the commit operation is denied
- developer sees a descriptive error message
- architecture team is notified of the failed commit
- developer gets email with further explanation
Custom checks at TCU

Brazilian Federal Court of Accounts (tcu.gov.br)
We’ve created 51 custom checks

• 27 checks for architecture conformance
• 14 checks for coding guidelines
• 10 checks for application security
Architecture conformance checks enforce for example

- the layered architecture
- placement of data access logic
- mandated generalizations
- naming of software elements
- design standards in reference architectures
Some lessons learned...
Add to pre-commit hooks only the checks
  • with zero violations in the target codebase
  • that enforce mandatory rules
But add *all* checks to sonarqube analysis
And run analysis in your continuous integration
Adopt an architecturally-evident coding style

An **architecturally-evident coding style** encourages you to embed hints in the source code that makes the architecture evident to a developer who reads the code.
Let developers know about the custom checks and suggest changes and improvements
Let architects understand the potential of the custom checks
For more information

Email: pmerson@acm.org


The single best thing you can do for the long term health, quality and maintainability of a non-trivial software system is to carefully manage and control the dependencies between its different elements and components by defining and enforcing an architectural blueprint over its lifetime

—Alexander Zitzewitz

Loving your architecture == defining and enforcing it
Backup slides
Some results...
750+ commit attempts denied in the first 24 months
Enhancements to the checkstyle and PMD APIs submitted to these open source projects
~12,000 violations fixed in the codebase

The real value was

• increased code quality
• improved awareness of the architecture and coding guidelines
More lessons learned...
Be careful not to become the “architecture police”
Be the “architecture mentors”
Create a mechanism to easily trump a check in exceptional cases
Automate the email with further clarification about a denied commit attempt