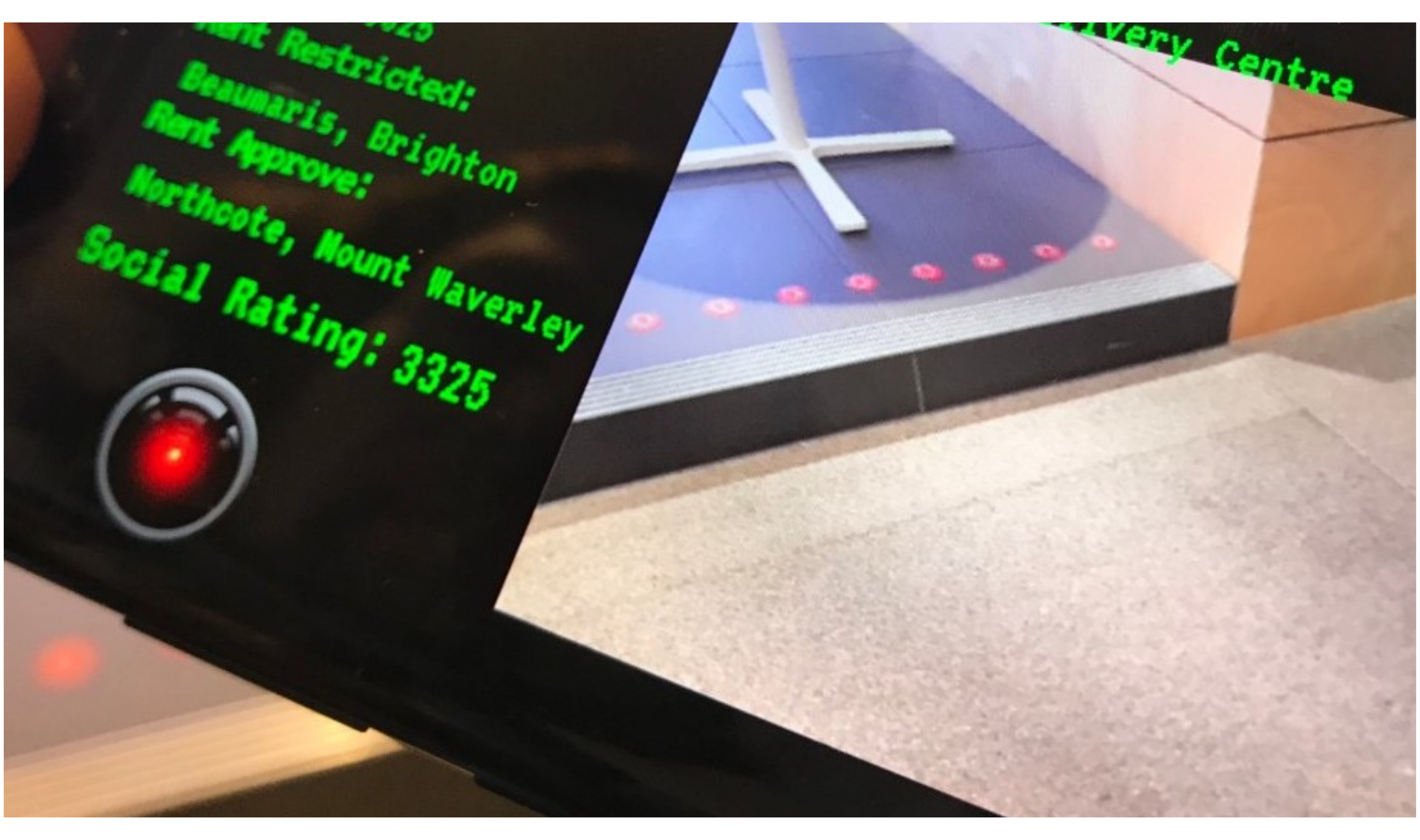
What good software looks like at REA

Alison Rosewarne
12th Jun 2018 — Read in 7 mins

Share



Introduction

Over the years we have created a lot of software at REA ranging from internal tools used by our customer experience team, to mobile apps used by millions of users each month, to data processing engines that crunch hundreds of gigabytes of data. While each piece of software has different functional and non-functional requirements we still have consistent expectations of that software as well as alignment on architectural principles. By taking this general consensus floating in the ether and capturing it using succinct language within high-level categories we have formed a foundation for discussion to ensure we can keep producing good software regardless of team, technology, and timeframe. Additionally, we routinely use this framework to assess all new and existing software to ensure our software continues to meet expectations and, where concerns are detected, to drive change.

The categories – overview

When considering the qualities of good software we use the following lenses:

- Development (aka "dev") – described as "Can I setup the codebase, understand it, and confidently make changes?"
- Operations (aka "ops") – described as "Can I deploy the system, understand it (and dependencies), handle DR [disaster recovery], and know if it is performing in line with established SLAs [Service Level Agreements]?"
- Architecture (aka "arch") – described as "Does the system encapsulate a single responsibility with a clearly defined interface within understood realms?"

For each lens we have identified specific criteria to assess whether the software:

- Meets our expectations
- Partially meets our expectations
- Falls significantly short of our expectations

We debated including "quality" and "security" separately but concluded that these cross cutting concerns were better handled within expectations across the three identified lenses. This contrasts with the architectural lens which, while also cross cutting, was specifically called out to encourage discussion.

These valued qualities are a product of the domain we operate within as well as our history, practices, and organisational structure. In short: our software (ranging in age from months to years old) needs to change frequently (ranging from every other month to multiple times a day) with changes made by a variety of people (of varying skill levels based both in Australia and China). Therefore, we value a low barrier for making changes (without compromising on quality) including the delivery of that change to the stakeholders. Additionally, our production environments are maintained by our teams around the clock therefore we value stability and repeatability of maintenance tasks. Finally, our architectural approach enables both concerns.

To fully understand the context requires understanding a fair bit about REA now and the journey we have taken.

About REA

We've been producing software at REA for over 20 years and over time much has changed.

Historically the language of choice was Perl. These days teams typically choose one of: Scala, Java, Ruby, or JavaScript for web/api development; Swift or Kotlin for mobile development; and Scala or Python for data pipeline development. Initially the development team could literally fit within a garage. Now our IT delivery team numbers in the hundreds, spanning multiple continents, with an agile way of working within a Spotify like structure. The production environment was once ad-hoc servers. This migrated to data centres and, ultimately, the cloud.

These large changes are all givens now. The most significant changes in recent times that more directly influence our consideration of good software are:

- adopting a micro-service architecture;
- embracing devops; and
- our application custodianship model.

The shift to a micro-service architecture rapidly gained traction as a means of consolidating a particular responsibility within a small and easily understood component that was verifiable and deployable without dependencies. This contrasted with the complex monolithic systems, each one developed by an individual non-cross functional team, that had accrued features and blurred domain boundaries over time and typically could only be tested (manually or automatically) with specific versions of their monolith friends.

In adopting this change, we've seen deep expertise in a single technology stack give way to polyglot programmers with broader knowledge, the variety of technical approaches (be it languages, frameworks, tooling) expanding, and the number of systems created increase dramatically.

A key enabler supporting the micro-service architecture has been a healthy devops culture. In the monolithic era a dedicated operations team deployed all software and dealt with maintenance and support of the staging and production environments around the clock. Today the teams that create the software are responsible for its lifecycle end to end including deployment, monitoring, and support (business and after hours, as required). Automation coupled with infrastructure as code have streamlined delivery.

And finally, the first two (coupled with organisational change over time) have contributed to our application custodianship model. Cross-functional teams at REA are organised into tribes that focus on specific internal and external segments of our business. These teams own all of the products and hence software within their domain from the monoliths of yesteryear to the plethora of micro-services created today. Teams own 'systems' as this includes software, data stores, CI servers, infrastructure components, etc.

Time is necessarily split between maintaining and improving the existing systems and creating new things, driven by value and risk. All systems are internally open source and other teams may submit changes through a pull request model. However, it is the application custodians who own the architecture, technical decisions, and quality of the systems hence they are consulted if another team wishes to make a major change and their approval is required for any change.

It is these custodians who perform quarterly review and assessment of the systems they own with reference to the criteria. The output is referred to as its health rating and feeds into a number of activities. Teams, application custodianship, and system health is tracked centrally and internally available in a simple web UI.

Dev

As defined above, the development lens relates to the changeability of the system. We need easily changeable software to support a rapid pace of features as well as quick response to defects or vulnerabilities.

We can say the following about systems that meet our expectations:

- Code is well factored and easy to understand.
- The development environment is easy to setup.
- Development feedback loop is short.
- CI [continuous integration] exists and executes automated tests that cover core functionality and enforce consumer contracts.
- Appropriate measures exist to protect customers, consumers, and data.

We can say the following about systems that partially meet our expectations:

- Automated tests require many external dependencies.
- Application custodian assigned but not performing the role.
- Dependent library versions not locked down.
- Build knowledge contained within CI server rather than source control.
- Design documentation and decision history doc (including language choice) is unavailable or not up to date.

We can say the following about systems that fall significantly short of our expectations:

- CI build is flakey.
- Based on unsupported language/OS/framework version (e.g. ruby 2.1)
- Unclear testing strategy.
- Not able to run application locally.
- Not able to run packaging or tests locally.

The systems created can only achieve these expectations where support is available for discussion (including disagreement), sharing, and training. Additionally, the barrier has been substantially lowered through centrally provided resources that all teams can leverage (such as github, wikis, artefact repositories, and CI servers).

Ops

As defined above, the operations lens relates to the reliability and maintenance of the production environment. We need performant systems to provide the best possible user experience and we need repeatable automated processes to reduce errors and waste.

We can say the following about systems that meet our expectations:

- Support doc exists (git or community [internal wiki]) and linked from catalogue.
- Monitoring and alerting as per established patterns and SLAs (logs in splunk).
- Data backed up regularly & securely as per CIA (Confidentiality, Integrity, Availability) pillars.
- Infrastructure as code, deployments automated, servers secured, secret information is protected, (cattle not pets).
- Production environment performs in line with clearly established SLAs.

We can say the following about systems that partially meets our expectations:

- Difficult to diagnose production issues.
- Infrequent scans for vulnerabilities / known issues (e.g. AWS Trusted Advisor, whitehat, tenable, etc).
- Lack of frequent patching.
- Inconsistent infrastructure across environments.
- Deployment requires extensive ops access and knowledge.

We can say the following about systems that falls significantly short of our expectations:

- Known critical or high impact vulnerabilities or insufficient access restrictions (e.g. firewall, network segregation, running as root).
- Inappropriate logging (either too much or not enough, refer [logging guidelines doc]).
- Cannot be forklifted into the cloud.
- Reliant on deprecated services.
- Excessive production alerts.

Once again, the openness to debate and commitment to sharing increases the likelihood that expectations will be met. Also, we have invested centrally in common tools and services (including vulnerability scanning, delivery engineering, deployment, monitoring, alerting, logging, backups, etc).

Arch

As defined above, the architecture lens relates to the responsibility of the system and its knowledge within a domain. Poorly architected systems carry a significant effort to change, cannot easily be migrated to the cloud, and provide a substantial barrier to innovation and experimentation.

We can say the following about systems that meet our expectations:

- Concerned with a single responsibility or business operation.
- Well encapsulated with a well-defined interface.
- Functionality gracefully degrades in the face of failure.
- Components are loosely coupled.
- Easy to understand system's place and purpose within the overall architecture.

We can say the following about systems that partially meet our expectations:

- Details of internal storage leak out through the interface.
- Terminology and data dealt with outside of application's bounded context.
- Too many dependencies on other components.
- Unnecessary runtime coupling.
- Duplicates responsibility of an existing system.

We can say the following about systems that fall significantly short of our expectations:

- Multiple systems writing data without a clear source of truth.
- Involved in a circular dependency relationship.
- Utilises shared logic via shared libraries.
- Poor/inappropriate abstraction applied to data or API access (too much or not enough).
- Inappropriate data retention implementation.

Once again, support is key to meeting expectations. We have a central architectural team available for consultation as well as peer review of all major technical choices.

Conclusion

The bullet points provided across the development, operations, and architectural lenses describe the kinds of things we look for in the software (and systems) we create as indicators of their quality based on our shared understanding. Rigorous and ongoing debate about what is good software is a key component of the craft which is encouraged. Periodic review of our catalogue keeps us honest about maintenance and trade-offs as well as feeding into most activities relevant for a digital business.

The ratings are available as an A3 canvas here.

Share

Alison Rosewarne
View profile

More from the blog

View all articles

TECH
1ST AUG 2023
Introducing Atomic (Part 2)

TECH
14TH JUL 2023
Unleashing Innovation: A Recap of the Global AI Hackathon in the USA!

TECH
6TH JUL 2023
Introducing Atomic – Pa

1995 2007 2014 2016 2017 2018 2020 2021 2022