**\*\* Have off-sites, team lunch, do hackathon, take a day off as a team.**

**Do good \*\***

[**Before Starting**](#_dezgbmd6rapw) **1**

[**UX**](#_syk68ct09j8e) **1**

[**Architecture**](#_sfpy1uhqsdmv) **2**

[**Development environment**](#_lmcropnd403z) **2**

[**Code repositories**](#_igtged4yw0c5) **2**

[**Coding convention**](#_731pmcmimby8) **2**

[**Security**](#_8jei41n6owo) **3**

[**Quality**](#_tvl858ugrxyc) **3**

[**Static code analysis**](#_cfwgaisuds9c) **4**

[**Development practices**](#_ivs6cyt13dt7) **4**

[**Continuous Integration**](#_smqtkkz063xg) **4**

[**Continuous Delivery/Deployment**](#_v0mkwarc3ztq) **4**

[**Data**](#_52dvp5ptie1r) **5**

[**Support - Monitoring**](#_w5qjm24foite) **5**

[**Support - Alerting**](#_8unxdls6x3d9) **5**

[**Resources**](#_envmx0se9mpz) **5**

#### 

#### Before Starting

* Understand the [Agile Principles](http://agilemanifesto.org/)
* Understand the [Lean Principles](https://www.lean.org/WhatsLean/Principles.cfm)
* Understand what [DevOps](https://aws.amazon.com/devops/what-is-devops/) is, and its [principles and practices](https://dzone.com/articles/devops-devops-principles)
* Understand what [MVP](http://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp) is
* Understand what [Product Ownership](https://www.youtube.com/watch?v=502ILHjX9EE) is (14:10 if your organisation doesn’t like truth and honesty, you won’t like Agile)
* Follow the [12-factor app](https://12factor.net) techniques.
* Understand the [Testing Pyramid](https://martinfowler.com/bliki/TestPyramid.html)
* Bake-in quality and security into your product
* **“You build it, you run it; you break it, you fix it”**

#### UX

* Make sure the UX is an integral part of your team
* Experiment. Experiment. Experiment
* Prototype. Prototype. Prototype.
* Get involved in ideation, user surveys, research and design facilitation
* Involve and experiment with real users
* Your customers and users are also your stakeholders
* Improve and grow your [Design Literacy](https://blogs.adobe.com/creativecloud/3-keys-to-improving-design-literacy-for-non-designers/)
* Collect data for insight on how successful the product actually is
* Usability makes or breaks your product

#### Architecture

* Follow the [Evolutionary Architecture](https://www.thoughtworks.com/radar/techniques/evolutionary-architecture) concept
* Follow the [Ten Commandments](https://centerbrook.com/blog/2010/03/the-ten-commandments-of-architecture/) of Architecture
* [Ivory-tower architecture](http://www.agilemodeling.com/essays/enterpriseModelingAntiPatterns.htm#IvoryTowerArchitecture) **doesn’t work**
* Do architecture often and in collaboration
* Remove hard-coupling and dependencies
  + OAS based APIs and microservices, consider using [Swagger](https://swagger.io/)
  + Contracts based development for both internal and external components - mock and stub as appropriate when unit testing
* [Microservice](https://martinfowler.com/articles/microservices.html) architecture [is not the silver bullet](https://martinfowler.com/articles/microservice-trade-offs.html)
* Consider other options e.g. [serverless](https://martinfowler.com/articles/serverless.html), PaaS
* [Design for failure](https://blog.risingstack.com/designing-microservices-architecture-for-failure/)
* Think self-monitoring, self-regulating and self-healing mechanisms
* Use the [Technology Radar](https://www.thoughtworks.com/radar) to guide your team’s architecture decisions
* “Can’t do architecture without programming, and can’t do programming without architecture” - [Martin Fowler](https://www.thoughtworks.com/talks/agile-architecture-rethink-2014)

#### Development environment

* Choose a well-known (and free) IDE (unless it’s .Net or C# - use Visual Studio)
* Documented, easy and quick to set up
* Repeatable, rebuildable everytime - no snowflake setup
* No special tweaks here or there ,use tools out-of-the-box and standardise
* Use automated build tool e.g. Maven or Cake and Fake

#### Code repositories

* Git preferred - github, gitlab or BitBucket
* [Learn](https://try.github.io/levels/1/challenges/1) git, follow [best practices](https://sethrobertson.github.io/GitBestPractices/)
* Understand how to use git, and understand how distributed repo works
* Implement [Versioning policy](http://semver.org/)
* Use [Trunk Based Development and Short-Lived Branches](https://barro.github.io/2016/02/a-succesful-git-branching-model-considered-harmful/) strategies

#### Coding convention

* [Conventions for different languages](https://en.wikipedia.org/wiki/Coding_conventions), Google’s [Java style](https://google.github.io/styleguide/javaguide.html)
* Conventions should be standardised across the organisation
* Follow the convention and include it in [code quality check](#_cfwgaisuds9c)
* Consistency in the codebase is a must; code individualisation is **unacceptable**
* Make sure everyone in the team understands and follows the agreed coding convention

#### Security

* Security engineer needs to be part of your team
* Check for vulnerabilities on own code and all dependencies
* 2nd Factor - on dependencies
* 3rd Factor - on configuration parameters
* Implement [OWASP Proactive Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls) (the list is ordered in importance)
* Self-service security, and make it **part** of the build: [snyk](https://snyk.io/), [Brakeman](https://brakemanscanner.org/), [dependency checks](https://www.owasp.org/index.php/OWASP_Dependency_Check)
* [Security scan](https://blog.docker.com/2016/05/docker-security-scanning/) your containers and do [dependency checking](#_4c8ru85sfv8x)
* Carry out targeted [DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) at Acceptance or UAT
* Automated functional and integration testing of security features
* Automated security attacks, using [Gauntlt](http://gauntlt.org/) or other tools
* Automated infrastructure security testing e.g. [InSpec](https://www.inspec.io/)
* Carry out targeted manual Penetration Testing

#### Quality

* [TDD](https://martinfowler.com/bliki/TestDrivenDevelopment.html) - you need to understand this
* Automate. Automate. Automate
* Follow the principles of [Testing Pyramid](https://martinfowler.com/bliki/TestPyramid.html)
  + Beware of the [Testing Cupcake](https://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern)
  + Determine the layers of the pyramid for your application - [start simple](https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html)



* For automated tests
  + [JavaScript](https://medium.com/powtoon-engineering/a-complete-guide-to-testing-javascript-in-2017-a217b4cd5a2a) - see [below](#_envmx0se9mpz) for list of frameworks
  + “Record and playback testing” is **unacceptable** as these will create non-deterministic tests.
  + Understand [Stubbing and Mocking](https://martinfowler.com/articles/mocksArentStubs.html)
  + BDD should be used ["in all the places where the business has reason to have opinions about the behaviour."](https://cucumber.io/blog/2016/07/20/where_should_you_use_bdd)
* Understand what [Test Coverage](https://martinfowler.com/bliki/TestCoverage.html) is, and determine the right coverage as part of your [static code analysis](#_cfwgaisuds9c)
* Never have any [non-deterministic test](https://martinfowler.com/articles/nonDeterminism.html)
* Test based on Risk Assessment
* Do load and performance testing
  + Consider the future load requirements of the application
  + [Gatling](http://gatling.io/), JMeter, SoapUI
* Do security and penetration [tests](#_4c8ru85sfv8x)
* Automate. Automate. Automate
* [TDD](https://martinfowler.com/bliki/TestDrivenDevelopment.html) - understood this now? If not - read about it again.

#### Static code analysis

* Use SonarQube or similar tools
* Include [code convention](#_731pmcmimby8) as part of the check
* Agree on the metrics your team should start with
* Monitored and adjusted as your team matures
* Include it in the CI pipeline and **fail** the build when your quality metrics drop

#### Development practices

* Always do peer review through Pull Requests mechanism
* Pair-programming could reduce the need for extensive peer-review
* Do documentation in Readme file in the repo
* [Self-documenting](https://www.martinfowler.com/bliki/CodeAsDocumentation.html) code is necessary
* Follow the Boy Scout rules - Clean Code book p14
* No Broken Windows - Clean Code book p8
* Understand what [Refactoring](https://www.agilealliance.org/glossary/refactoring/#q=~(filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'refactoring))~searchTerm~'~sort~false~sortDirection~'asc~page~1)) is and Just Do It
* Understand what [Technical Debt](https://martinfowler.com/bliki/TechnicalDebt.html) is, and apply when appropriate
* Alignment to architectural and technology governance
* “Keep the Codebase Healthy” - [Martin Fowler](https://www.thoughtworks.com/talks/agile-architecture-rethink-2014)

#### Continuous Integration

* Automated build and notifications
* When a build should be activated, at every commit?
* Build should stop at ANY test failure - no excuse
* Build should stop at a predetermined decrease of code quality
* Roll your own, Jenkins, Hudson, Octopus etc.
* SaaS: Codeship, Travis CI, Bamboo, Hudson, GoCD

#### Continuous Delivery/Deployment

* [Infrastructure as Code](https://www.thoughtworks.com/insights/blog/infrastructure-code-reason-smile), include unit tests!
* Automatically build the release package
* Automatically deploy to intended environments
* Release package will be promoted to the next stage only on successful build
* Consistent server build and environment, no snowflake or pets server or environment
* 10th Factor - all environments should be very similar, dev, test, UAT, QA, prod.
* Utilise containers i.e. Docker
* Utilise an orchestration manager e.g. Kubernetes, Docker Swarm, Apache Mesos
* Utilise a configuration manager, Infrastructure as Code, e.g. Anslble, Chef, Puppet, even on PaaS
* Consider the right deployment technique:
  + A/B testing
  + Red/Blue deployment
  + Canary
  + Feature toggles
  + Dark launches
* [Decouple deployment from release](https://www.thoughtworks.com/radar/techniques/decoupling-deployment-from-release)

#### Data

* Based on the organisation’s Data Strategy
* Build in metrics to provide data to the business and for health monitoring
* Consider support scenario
* Use data to make your life easier
* Use data to help your team to make decisions

**\*\* The aim below is to not have anyone got woken up at 3am by your bugs.**

**You break it, you fix it \*\***

#### Support - Monitoring

* Consider the principle of Least Privilege for access control
* Consider support requirements
* Put monitoring everywhere, prioritise alerting on component’s importance, resiliency, set-up, backup etc.
* New Relic, OMS, Splunk, ELK, Raygun
* Build a dashboard which is useful at all levels, consider Splunk, Grafana

#### Support - Alerting

* Consider the application’s SLA
* Consider the support agreements
* Consider Segregation of Duties
* Determine the metrics that will trigger alerts
* Alert types e.g. Info, Warning etc.
* Select appropriate tools to make your life easy: IM, SMS, email, flashing lights, sirens etc.

Remember self-monitoring, self-regulating and self-healing?

===============================================================

#### Resources

* <https://guide.freecodecamp.org/>
* <https://github.com/mr-mig/every-programmer-should-know>
* <https://stevewedig.com/2014/02/03/software-developers-reading-list/>
* [https://martinfowler.com](https://martinfowler.com/bliki/GivenWhenThen.html)
* <https://www.thoughtworks.com>
* [Technology radar](https://www.thoughtworks.com/radar) - in conjunction with Gartner’s material
* Beware of framework/language/technology’s Hype Cycle
* <https://trello.com/b/lw96ei6d/books>
* <https://www.safaribooksonline.com/library/view/lean-ux-2nd/9781491953594/>
* DevOpsSec - <http://www.oreilly.com/webops-perf/free/files/devopssec.pdf>
* JavaScript Testing
  + Sinon - mocking and stubbing
  + Protactor - end to end tests (use with Selenium)
  + Jasmine framework
  + Mocha framework
  + Chai - assertion library
  + Karma test runner - run tests in browsers - can run in BrowserStack or SauceLabs
  + Protractor for Angular (end-to-end test framework)
  + [Nightwatch](http://nightwatchjs.org/) end-to-end for automated browser based apps
* AWS Device Farm alongside BrowserStack and SauceLabs
* PaaS
* VSTS - Azure App Services
* CodeStar - AWS
* OpenShift - RedHat
* Tools
* Jira
* Wiki
* Chat/IM - slack, hipchat