I. Code

1. Master branch must be read-only. No exceptions

2. Each Release to Production must be done by creating the new branch from the master with version number 'prod-v.1.1', 'prod-v.1.2' etc

3. Every idea (minor, major, bug fix, feature, whatever) starts from ticket creating '#105 fix' etc

4. Each amendment should begin from creating new branch dev-{feature name}

5. Each pull request should contain a link to the corresponding ticket number.

6. Each branch after merging with master should be deleted to avoid any code duplication

7. Error reporting or Improvement suggestion

8. Code style. Class / Method / Function / Variable / Constant naming, verbs vs nouns, a compound name is a big trouble

9. Code static analysis

10. Code review

11. Scope of change

12. PDD. Cutting edges with TODO and interfaces

II. Databases

1. SQL scripts for database creating must be in the central repository with the code.

2. Each amendment to SQL script should be done via 'delta SQL scripts' to trace changes

III. Deployment

1. Deployment cycle as short as possible

2. Continuous Integration

3. Continuous Delivery

4. CDN

5. The code begins to bring money only after delivery

IV. Backup / restore

1. All backups should be automated

2. All backups operations should be done by pressing one key / running one command

3. All backups should be tested to the ability to restore

4. All restore operations should be done by pressing one key / running one command

5. Incremental backups (Day / Month / Year)

V. Environment Details

1. Environment settings must be stored in the central repository.

2. The environment should be able to reproduce in one command/click

3. The DEV / PROD / QA environments should be same.

VI. Configuration Issues, Nuances

VII. Dependencies

1. List of dependencies.

2. Notes about compatibility

3. Notes about dependency licenses

4. Notes about dependency locations

5. Testing and upgrading dependencies

VIII. Documentation

1. All documentation should be stored in the central repository

2. Documentation should cover at possible functions and application behavior

3. No communication

4. Glossaries

IX. Tests

1. Test coverage 70-80%

2. Unit tests 90%+

3. A/B tests

3. Integration tests

4. Mockups

5. Regress testing

6. Thread safety

7. Single statement unit tests

8. The unlimited number of bugs. When to stop testing?

9. TDD

10. Testing vs. Debugging

X. Metrics

1. Accounting resource-consumption tasks

2. Accounting module usage statistics

XI. Security

1. Logging and analyzing IP-addresses

2. Logging and analyzing users activity

XII. Architecture

1. Scalability

2. Latency

3. Throughput

4. Code decoupling

5. Repository decoupling

6. NO Antipatterns: https://www.yegor256.com/2016/02/03/design-patterns-and-anti-patterns.html

7. Maintainability

8. Error processing

9. Synchronized vs. non-locking

10. Gradients of immutability https://www.yegor256.com/2016/09/07/gradients-of-immutability.html

11. Where to validate? In the constructor or the decorator?

12. Coupling vs. cohesion

13. Throwing an Exception Without Proper Context

14. Robust vs. Fragile

15. One vs. multiple returns

16. Hacker vs. designer

XIII. Development Process

1. No informal discussions.

2. All information, all the knowledge should store with the project

3. First code duplication should lead to creating a new module/ class/function

4.Speed vs. Quality

5. Constant improving and constant learning

6. Task isolation

7. Responsibility borders

8. No bureaucracy

9. All processes should be described well

10. Definition of Done

11. Improve anything and meticulously

12. 1.01 ^ 365= 37

XIV. Specification

https://www.yegor256.com/2015/11/10/ten-mistakes-in-specs.html

1. No Glossary or a Messy One

2. Questions, Discussions, Suggestions, Opinions

3. Mixing Functional and Quality Requirements

4. Mixing Requirements and Supplementary Docs

5. Un-measurable Quality Requirements

6. Implementation Instructions

7. Lack of Actor Perspective

8. Noise

9. Will Work, Needs to Work, Must Work

10.

XV. Management

1. Scope.

2. Issues.

3. Risks

XVI. Seven Deadly Sins

1. Anti-Patterns

2. Untraceable Changes

2.1. Always Use Tickets

2.2. Reference Tickets in Commits

2.3. Don't Delete Anything

3. Ad Hoc Releases

4. Volunteer Static Analysis

5. Unknown Test Coverage

6. Nonstop Development

7. Undocumented Interfaces

XVII. Twelve Mistakes in Agile Manifesto

1. Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.

2. "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

3. "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."

4. "Business people and developers must work together daily throughout the project."

5. "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."

6. "The most efficient and effective method of conveying information to and within a development team is the face-to-face conversation."

7. "Working software is the primary measure of progress."

8. "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."

9. "Continuous attention to technical excellence and good design enhances agility."

10. "Simplicity—the art of maximizing the amount of work not done—is essential."

11. "The best architectures, requirements, and designs emerge from self-organizing teams."

12. "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."