Einleitung

Ziel dieses Dokuments ist es langjährige Erfahrung aus der Wartung und Weiterentwicklung von Software zu sammeln und die daraus abgeleiteten Erkenntnisse in neue Projekte zurückzuspielen.

Mit dem Ziel der fortwährenden Professionalisierung und Anpassung der Softwareentwicklungsprozesse an die steigende Projektgröße, die auch eine höhere Komplexität mit sich bringt, formuliert dieses Dokument Anforderungen an die zu erstellende Software.

Dabei gilt der gesunde Menschenverstand und nicht in jedem Projekt ist jeder Punkt verpflichtend – aber man sollte sinnvoll drüber nachgedacht und bewusst entschieden haben.

Und wenn am Ende dieses Prozesses, dann nur ein Teil der Anforderungen umgesetzt ist, so sollte man nochmal über seinen eigenen Anspruch an die Softwareentwicklung nachdenken – die Liste besteht nämlich eigentlich aus Selbstverständlichkeiten und die einzelnen Punkte sind „state-of-the-art“ in professioneller Softwareentwicklung.

Inhalt

1. Dokumentation

1.1 Anforderungen

Die funktionalen und nicht-funktionalen Anforderungen an die Software sind spezifiziert. Die Spezifikation enthält mindestens die folgenden Artefakte:

- Liste der Anwendungsfälle inkl. mindestens je 2-3 Sätzen Beschreibung. Wenn die Anwendungsfälle nur als Tickets vorliegen, sollte trotzdem eine Liste als Dokument gepflegt sein.
- Erwartetes Nutzeraufkommen im produktiven Betrieb
- Einzuhaltende Antwortzeiten
- Erwartetes Datenaufkommen im produktiven Betrieb (Mengenbetrachtung)
- Liste der einzuhaltenden Standards/Verordnungen etc. (z.B. BITV 2.0 für Barrierefreiheit)

1.2 Architektur

Die Architektur der Software ist klar und verständlich dokumentiert und aktuell. In der Architekturdokumentation sind mindestens die folgenden Artefakte vorhanden:

- Kontextdiagramm
- Bausteinsicht(en)
- Erläuterung des verwendeten Architekturmusters
- Liste der verwendeten Technologien
- Architekturentscheidungen; insbesondere, wenn kein Standard-Stack verwendet wird
- Liste der technischen Schulden
- Liste aller vorgenommenen Modifikationen von eingesetzten Bibliotheken und Frameworks

1.3 Entwicklerdokumentation

Das Entwicklungsvorgehen ist dokumentiert. Die Entwicklerdokumentation enthält mindestens die folgenden Artefakte:

- Anleitung zur Einrichtung der Entwicklungsumgebung (Docker, IDE, VMs…)
- Liste der verwendeten Entwicklungswerkzeuge (Tools, Plugins etc.)
- Entwicklungsvorgaben
- Code Conventions
- Definition of Done
- Erläuterung des verwendeten Brachmodells und –prozesses
- Beschreibung des Reviewprozesses
- Workflows
- Anleitungen für Build, Deployment und Release der Software
- Beschreibung der Testumgebungen (-Stages) inkl. Konfiguration

1.4 Betriebshandbuch

Für den Betrieb der Software ist eine entsprechende Dokumentation vorhanden. Das Betriebshandbuch beinhaltet mindestens die folgenden Artefakte:

- Anleitung zu Installation und/oder Deployment neuer Releases oder Updates
- Beschreibung der einzelnen Server/Komponenten inkl. Mindestvoraussetzungen
- Liste der anwendungsspezifischen Konfigurationsparameter inkl. Beschreibung und erlaubter Werte
- Beschreibung, wie das Monitoring der Software erfolgt (technisch & fachlich)

2. Tests

2.1 Unit-Tests

Es sind automatisierte Unit-Tests vorhanden (auch für JavaScript etc.). Die Codeabdeckung der Unitestes erreicht eine Zweigabdeckung von mindestens 80%.

2.2 Integrationstests

Es sind automatisierte Integrationstests vorhanden (z.B. via SoapUI, Postman, Selenium, Cypress etc.). Es wird eine ABC-Einteilung der Geschäftsprozesse vorgenommen.

Kategorie A umfasst alle geschäftskritischen Kernprozesse der Anwendung. Sie sind dauerhaft in verwenden und dürfen nicht ausfallen.

Kategorie B umfasst alle Standardprozesse der Anwendung. Sie werden regelmäßig verwendet und führen zu einer starken Beeinträchtigung, wenn sie ausfallen.

Kategorie C umfasst alle sonstigen Prozesse der Anwendung. Sie werden selten verwendet und/oder führen nur zu einer geringen Beeinträchtigung, wenn sie ausfallen.

Die drei Kategorien werden wie folgt durch Integrationstests abgedeckt:

- Mindestens 80% der A-Prozesse sind jeweils mit einem Test für den Positivfall und zwei Tests für Negativfälle abgedeckt.
- Mindestens 70% der B-Prozesse sind jeweils mit einem Test für den Positivfall und einem Test für Negativfälle abgedeckt.
- Mindestens 50% der C-Prozesse sind jeweils mit einem Test für den Positivfall abgedeckt.

2.3 Last- und Performancetests

Es werden automatisiert Last- und Performancetests durchgeführt, im besten Fall auf einer produktionsnah konfigurierten Testumgebung. Für die Tests werden die in den nicht funktionalen Anforderungen vorgegebenen maximalen Nutzerzahlen und Datenvolumen verwendet. Die Antwortzeiten der Software bleiben in den vorgegebenen Werten.

Die Last- und Performancetests werden regelmäßig durchgeführt. Je nach Projektumfang erfolgt dies mindestens einmal alle 1 – 3 Monate. Die Ergebnisse und ggf. daraus resultierende Maßnahmen werden dokumentiert.

2.4 Massendatentests

Es werden automatisierte Massendatentests durchgeführt, im besten Fall auf einer produktionsnah konfigurierten Testumgebung. Für die Tests wird das in den nicht funktionalen Anforderungen vorgegebene maximale Datenaufkommen verwendet. Die Tests beinhalten die Ausführung von allen vorhandenen datenintensiven Anwendungsfällen (z.B. Massendatenimport und –export, Statistikberechnungen etc.). Die Auswertung der Massendatentests trifft Aussagen über die folgenden Themen:

- Speicherauslastung
- Garbage-Collection
- Auswirkung auf die Performance
- Auswirkung auf die Datenbank (z.B. arbeiten Views/Queries weiterhin performant?)

Die Massendatentests werden regelmäßig durchgeführt. Je nach Projektumfang erfolgt dies mindestens einmal alle 1 – 3 Monate. Die Ergebnisse und ggf. daraus resultierende Maßnahmen werden dokumentiert.

2.5 Automatisierung der Testumgebungen

Die Erzeugung und Konfiguration der Test-Stages soll möglichst automatisiert erfolgen z.B. durch den Einsatz von Docker, Ansible etc.

2.6 Produktionsnahe Testumgebung

Mindestens eine Testumgebung ist der geplanten bzw. bereits existierenden produktiven Umgebung so nah wie möglich nachempfunden.

Auf dieser produktionsnahen Testumgebung führt das QS-Team den Test (spätestens zur Abnahme) der Software durch.

3. Code-Qualität

3.1 SonarQube

Die Software wird regelmäßig bei jedem Push in das Repository mit SonarQube analysiert. Es müssen die folgenden Grenzwerte eingehalten werden.

Bugs	= 0
Vulnerabilities	= 0
Code Smells Blocker	= 0
Code Smells Critical	= 0
Code Duplication	< 5%

Ausnahmen müssen im SonarQube nachvollziehbar begründet sein!

Grundsätzlich muss das für das Projekt eingerichtete QualityGate eingehalten werden.

4. Continuous Integration

4.1 Build-Server

Es existiert für das Projekt ein Bereich auf dem (zentralen) Build-Server das regelmäßig mindestens die folgenden Aufgaben ausführt:

- Build der Software inkl. Ausführung Unit-Tests und SonarQube-Analyse bei jedem Push in das Repository
- Deployment auf Teststages
- Security-Check der verwendeten Bibliotheken und Frameworks (z.B. OWASP-Dependency-Check) mindestens einmal täglich
- Tägliche Ausführung der Integrationstests
- Ausführung der Last- und Performancetests
- Ausführung von Massendatentests

4.2 Artefakt-Repository

Auslieferungsartefakte und Release-Kandidaten sind nachvollziehbar und versioniert archiviert, z.B. in einem Nexus oder Artifactory.