Curriculum für

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Modul:
IMPROVE

Evolution und Verbesserung von Software-Architekturen

Version 1.6 (November 2015)

Inhaltsverzeichnis

Verzeichnis der Lernziele

0 Einleitung

0.1 Was vermittelt das Modul "IMPROVE"?

Teilnehmer lernen hier, Softwaresysteme und -architekturen anhand ökonomischer und technischer Ziele systematisch und methodisch zu verbessern. Dazu vermitteln entsprechende Schulungen die systematische Trennung von Problemen und Lösungsansätzen, das Erarbeiten von kurz-/mittel- und langfristigen Lösungsstrategien sowie deren Abgleich mit betriebswirtschaftlichen Zielen und Größen.

Zusätzlich zeigt der IMPROVE-Lehrplan typische Ansätze für Verbesserungen auf, beispielsweise Restrukturierungen und Refactoring, Verbesserungen der Analysierbarkeit, Prozessverbesserung, Verbesserung an technischer Infrastruktur, Verbesserung von Qualitätseigenschaften, etc.

0.2 Was vermittelt ein Advanced-Level-Modul?

- Der iSAQB-Advanced-Level bietet eine modulare Ausbildung in drei Kompetenzbereichen mit flexibel gestaltbaren Ausbildungswegen. Er berücksichtigt individuelle Neigungen und Schwerpunkte.
- Die Zertifizierung erfolgt als Hausarbeit. Die Bewertung und mündliche Prüfung wird durch vom iSAQB benannte Experten vorgenommen. Details im Web unter http://isaqb.org.

0.3 Was können Absolventen des Advanced Level (CPSA-A)?

CPSA-A-Absolventen können:

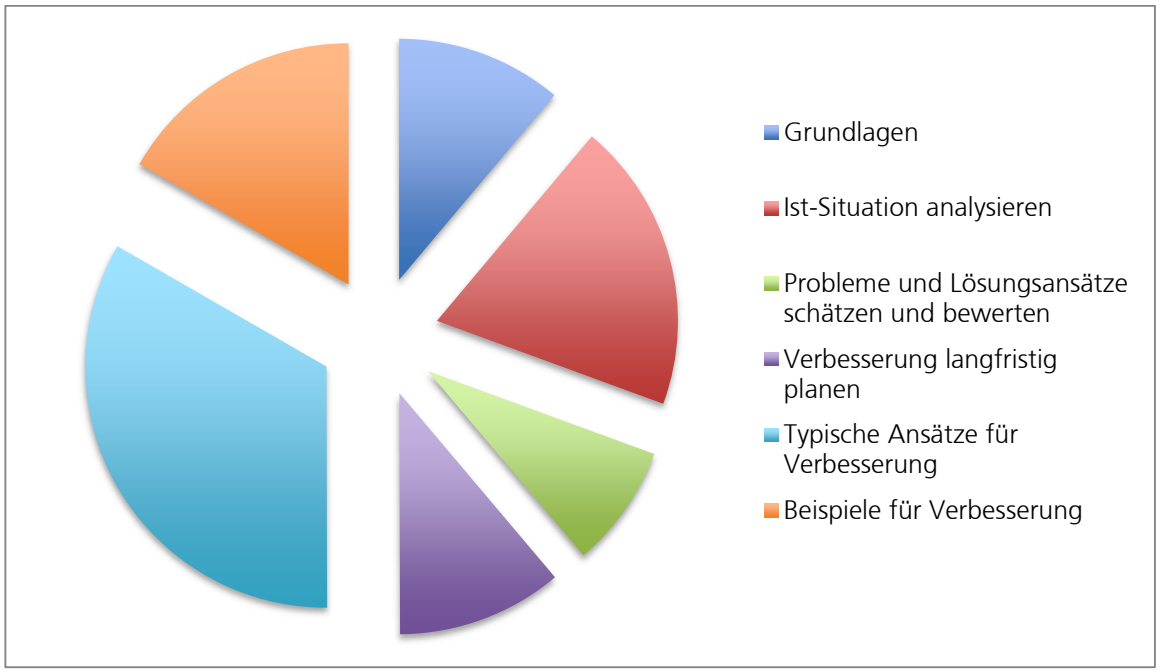
- Eigenständig und methodisch fundiert mittlere bis große IT-Systeme entwerfen.
- In IT-Systemen mittlerer bis hoher Kritikalität technische und inhaltliche Verantwortung übernehmen.
- Maßnahmen zur Erreichung und Verbesserung nichtfunktionaler Anforderungen konzeptionieren, entwerfen und dokumentieren. Sie können Entwicklungsteams bei der Umsetzung dieser Maßnahmen begleiten.
- Architekturrelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen.

0.4 Voraussetzungen zur CPSA-Advanced-Zertifizierung

- Eine erfolgreiche Ausbildung und Zertifizierung zum CPSA-F (Certified Professional for Software Architecture, Foundation Level).
- Mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche, dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen.
    - Ausnahmen auf Antrag zulässig (etwa: Mitarbeit in OpenSource-Projekten).
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit Points aus allen drei unterschiedlichen Kompetenzbereichen (Details siehe iSAQB-Website).
- Erfolgreiche Bearbeitung der CPSA-A-Zertifizierungsprüfung.

0.5 Gliederung des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer
Grundlagen	120 min
Ist-Situation analysieren	210 min
Probleme und Lösungsansätze schätzen	90 min
Verbesserung langfristig planen	120 min
Typische Ansätze für Verbesserung	360 min
Beispiele für Verbesserung	180 min
Gesamt (3 Tage à 360min)	1.080 min = 18h



0.6 Dauer, Didaktik und weitere Details

Die genannten Zeiten für die einzelnen Teile des Lehrplans stellen lediglich Empfehlungen dar. Die Dauer einer Schulung zu IMPROVE sollte mindestens 3 Tage betragen, kann aber durchaus länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

Lizensierte Schulungen zum IMPROVE-Modul tragen bezüglich der Zulassung zur abschliessenden Advanced-Level-Zertifizierungsprüfung folgende Punkte (Credit Points) bei:

Methodische Kompetenz:	20 Punkte
Technische Kompetenz:	10 Punkte
Kommunikative Kompetenz:	0 Punkte

0.7 Voraussetzungen für das Modul IMPROVE

Teilnehmer **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Praktische Erfahrung in Entwurf und Entwicklung kleiner bis mittelgroßer Softwaresysteme.
- Erste praktische Erfahrung in Wartung oder Weiterentwicklung von Softwaresystemen.
- Erste praktische Erfahrung im Umgang mit bestehendem Quellcode mittlerer oder großer Systeme, insbesondere Analyse, Bewertung, Qualitätsprüfung, Refactoring sowie Anwendung von Metriken.

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Kenntnisse und praktische Erfahrung im Refactoring (etwa gemäß [Fowler+99])
- Kenntnisse oder erste praktische Erfahrung in Review oder Bewertung von Software:
  - Grundkenntnisse in Software-Metriken, z. B. Metriken zur Messung von Kopplung, Kohäsion, Abhängigkeit, Komplexität
  - Grundkenntnisse der Laufzeitanalyse von Software, etwa Profiling, Tracing, Log-Analyse, Datenanalyse
  - Grundkenntnisse und erste Erfahrung in der Aufwandsschätzung von Entwicklungsaufgaben
- Erste praktische Erfahrung in der Anforderungsanalyse sowie dem Umgang mit unterschiedlichen Stakeholdern von Entwicklungsprojekten.
- Erste praktische Erfahrung in der Analyse von Entwicklungsprozessen.

0.8 Aufbau der Lerneinheiten aus Lernzielen

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

- **Begriffe/Konzepte**: Wesentliche Kernbegriffe dieses Themas.
- **Unterrichts-/Übungszeit**: Legt die Unterrichts- und Übungszeit fest, die für dieses Thema bzw. dessen Übung in einer akkreditierten Schulung mindestens aufgewendet werden muss.
- **Lernziele:** Beschreiben die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte.

Lernziele skizzieren auch die zu erwerbenden Kenntnisse und Fähigkeiten in entsprechenden Schulungen. Sie differenzieren in folgende (Relevanz)-Kategorien:

- (R1) Was sollen die Teilnehmer können? Diese Inhalte sollen die Teilnehmer nach der Schulung selbständig anwenden können. Innerhalb von Schulungen werden diese Inhalte unterrichtet und sollten zusätzlich durch Übungen oder Diskussion vertieft werden.
- (R2) Was sollen die Teilnehmer verstehen? Diese Inhalte werden in Schulungen thematisiert und können durch Übungen oder Diskussion unterstützt werden.
- (R3) Was sollen die Teilnehmer kennen? Diese Inhalte (Begriffe, Konzepte, Methoden, Praktiken oder Ähnliches) können das Verständnis unterstützen oder das Thema motivieren. Diese Inhalte werden in Schulungen bei Bedarf thematisiert, aber nicht notwendigerweise ausführlich unterrichtet.

Bei Bedarf enthalten die Lernziele Verweise auf weiterführende Literatur, Standards oder andere Quellen.

1 Grundlagen von Evolution und Verbesserung von Software-Architekturen

Dauer: 120 min

1.1 Begriffe und Konzepte

- Software-Architektur: Struktur, Bausteine/Komponenten, Schnittstellen, übergreifende (querschnittliche) Konzepte
- Veränderung, Evolution, Wartung, Verbesserung von Software
- Kategorien von Problemen und Risiken von Software (Technische Schulden)
- Kernbegriffe rund um Verbesserung und Änderung von Software
- Technische Pflege.

1.2 Lernziele

LZ 1-1: Gründe für Veränderungen an Software kennen und erläutern (R1)

- Funktionale Erweiterung oder Änderung
- Änderung von Qualitätsanforderungen oder -zielen
- Änderungen in technischem oder fachlichen Kontext (z. B. Änderung an externen Schnittstellen)
- Fehlerbehebung
- Organisatorische Änderungen (z. B. Änderungen juristischer Rahmenbedingungen oder Anforderungen, an Organisationsstrukturen in Fachbereichen, Entwicklung oder Betrieb)
- Kosten- oder Aufwandsreduktion, insbesondere bezüglich
  - Entwicklungskosten oder -aufwände
  - Betriebskosten oder -aufwände
  - Kosten der Fehlerbehebung und Kosten von Folgeschäden
  - Kosten beteiligter Prozesse
  - Opportunitätskosten
- Intrinsische Motivation beteiligter Personen, insbesondere Software-Entwickler und -Architekten
- Aktualisierung verwendeter Technologien, etwa Betriebssysteme, Middleware, Bibliotheken, Frameworks, Hardware oder ähnliche.

LZ 1-2: Typische Kategorien von Problemen an Software kennen und erläutern (R1)

- Konzeptionelle Schwächen, beispielsweise Verletzung konzeptioneller Integrität, Redundanz, Verwendung nicht angemessener Technologien, falsche Verwendung von Technologien

- Strukturelle Probleme in Daten oder Datenstrukturen
- Strukturelle Probleme im Quellcode, etwa:
  - fehlende oder unzureichende Modularisierung,
  - enge Kopplung,
  - hohe Komplexität,
  - geringe Kohäsion,
  - handwerkliche Schwächen in der Programmierung, etwa nicht-idiomatische Verwendung von Programmiersprachen oder –werkzeugen.
- Laufzeitprobleme in Systemen, etwa Instabilität, mangelnde Robustheit oder Zuverlässigkeit, mangelnde Performance oder zu hoher Ressourcenbedarf, mangelnde Skalierbarkeit, mangelnde Transparenz von Systemabläufen
- „Altlasten" durch häufige Anforderungsänderung unter Zeit- und/oder Kostendruck
- Verschiedene Arten von technischen Schulden erkennen und benennen
- Probleme bei Erreichung von Qualitätszielen oder -anforderungen, etwa mangelnde Änderbarkeit oder Wartbarkeit, geringe Flexibilität, mangelnde Sicherheit oder mangelnde Portabilität
- Probleme in Entwicklungs- oder Betriebsprozesse oder sonstiger beteiligter Prozesse (z. B. Anforderungsmanagement, Test/QS, Konfiguration, Deployment, Monitoring/Alerting).

LZ 1-3: Kernbegriffe für Evolution und Änderung kennen und erläutern (R1)

Kernbegriffe von Evolution und Änderung kennen:

- Problem (Issue)
- Lösungsansatz (Opportunity for Improvement)
- Kosten (von Problemen, Lösungsansätzen, Maßnahmen)
- Ursachen (Root cause) versus Symptom
- Risiko.

LZ 1-4: Mögliche Vorgehensweisen bei Änderungen kennen und erläutern (R2)

Beispielsweise:

- Ad-hoc Verbesserung, einmalige Verbesserung
- Schrittweise Verbesserung (mittel- bis langfristig angelegt)
- Verbesserung durch Neuentwicklung (Rewrite) eines Systems oder Teilen davon
- Rein strukturelle Verbesserungen (Refactoring)
- Konzeptionelle/strukturelle Verbesserung (Re-Architecting, Reengineering)

2 Ist-Situation analysieren

Dauer: 210 min

2.1 Begriffe und Konzepte

- Ist-Analyse, Stärken/Schwächen-Analyse
- Stakeholder
- Problem, Ursache versus Symptom
- Lösungsansatz

2.2 Lernziele

LZ 2-1: Grundlagen der Analyse zur Trennung von „Problem“ und „Lösung“ kennen und anwenden können (R1)

- Problemanalyse begrifflich von Problemlösung trennen
- Problem und Lösungsansatz in m:n Verhältnis einander zuordnen
- Komplexe Probleme in Teilprobleme zerlegen

LZ 2-2: Typische Praktiken und Methoden zur Ist-Analyse kennen und anwenden können (R1)

Typische Praktiken der Ist-Analyse kennen und für die jeweilige Situation, das Budget, die Zeit bzw. die Beteiligten angemessene auswählen können. Dazu gehören insbesondere folgende Ansätze:

- Stakeholder-Analyse und -interview
- Kontextanalyse
- Qualitative Analyse (etwa ATAM), insbesondere die Analyse von Qualitätszielen
- Analyse von strukturellen Abweichungen zwischen Soll- und Ist-Architektur
- Quantitative Analysemethoden und -praktiken, etwa:
  - Code-Metriken (Größenmaße, Kopplung, Komplexität, Kohäsion)
  - Organisatorische Metriken wie Kosten, Zeit und Anzahl. Beispiele: Fehleranzahl und Fehlerraten, Entwicklungsgeschwindigkeit, Kosten pro Feature, Kosten pro behobenem Fehler etc.
  - Laufzeitmetriken, wie Zeit- und Ressourcenbedarf und Werkzeuge zu deren Erhebung
- Datenanalyse: Untersuchung von Datenstrukturen und -inhalten
- Dokumentationsanalyse
- Analyse der technischen Umgebung (Laufzeit- und Betriebsumgebung: Hardware, Betriebsumgebung, Netzwerke, beteiligte Betriebssysteme und Infrastruktursoftware)

- Analyse von Entwicklungsprozessen (Anforderungsmanagement, Entwurf/Implementierung, Test/Qualitätssicherung sowie Übergabe an Betrieb/Produktion)
- Analyse von Betriebsprozessen
- Analyse weiterer Prozesse und Aufgaben, die Probleme für Entwicklung und Änderung von Systemen beeinflussen können, beispielsweise Management, Budgetierung, Support, Governance, Outsourcing/Offshoring, Lieferantenmanagement u. a.

LZ 2-3: Gefundene Probleme und Risiken methodisch dokumentieren können (R1)

- Teilnehmer sollen in der Lage sein, im Rahmen von Verbesserungs- und Veränderungsprozessen eine angemessene Dokumentation gefundener Probleme (Issues) und Risiken aufzusetzen
- Werkzeuge und Beispiele für Problemdokumentation

LZ 2-4: Stakeholderanalyse und -interviews planen und durchführen können (R1)

- Stakeholderanalyse zur Identifikation der wesentlichen beteiligten Personen sowie deren Rollen und Intentionen planen, durchführen und dokumentieren
- Methodiken zur Konzeption und Durchführung von Stakeholder-Interviews kennen
- Vorbereitende Fragebögen erstellen können
- Situativ im Verlauf von Interviews spezifisch auf sich ergebende neue Sachverhalte eingehen und diese in die Analyse einbeziehen

LZ 2-5: Kontextanalyse durchführen können (R1)

- Kontextabgrenzung durchführen und dokumentieren: Systeme bezüglich ihrer fachlichen und technischen Nachbarn abgrenzen und dabei externe Schnittstellen identifizieren.
- Zusammenhänge zwischen externen Schnittstellen und möglichen Stakeholdern erkennen und für Problemanalyse ausnutzen können.
- Probleme und Risiken externer Schnittstellen erarbeiten (u. a. über Interviews, Analyse bekannter Fehler, Laufzeitanalyse, Protokoll- oder Loganalyse, Analyse der organisatorischen Beziehungen, Analyse der Qualitätsmerkmale und/oder Service-Levels).

LZ 2-6: Code- und Strukturanalyse durchführen können (R1)

- (Statische) Analyse von bestehendem Quellcode und dessen Strukturen durchführen und dokumentieren. (Hierzu können in der Schulung entsprechende Werkzeuge eingesetzt werden, sind jedoch nicht Voraussetzung).

LZ 2-7: Grundlegende Laufzeitanalysen planen und durchführen können (R2)

- (Dynamische) Analyse von bestehenden Systemen planen und durchführen können, beispielsweise hinsichtlich Laufzeiten, Ressourcen-Bedarf, Lastverhalten. (Hierzu können in der Schulung Werkzeuge eingesetzt werden, sind jedoch nicht Voraussetzung).

3 Probleme und Lösungsansätze schätzen und bewerten

Dauer: 90 min

3.1 Begriffe und Konzepte

- Aufwand, Kosten, Schätzung, Beobachtung/Messung, Annahmen,
- betriebswirtschaftliche Größen
  - Investition, Ertrag, Kosten, Wert
  - Return-on-Invest (ROI)
  - Break-Even,
  - RTC- und BTC-Kosten
- Intervallschätzung, Gesetz-der-großen-Zahl.

3.2 Lernziele

LZ 3-1: Betriebswirtschaftliche Größen kennen und erläutern (R1)

- Kosten und Wert als Begriffspaare erläutern
- Unterschiedliche Arten von betriebswirtschaftlichen Größen benennen und einordnen können, beispielsweise:
  - Kosten für direkte Aufwände, Ausbildungskosten, Kapitalkosten (direkte und indirekte Kosten der Investition), Betriebskosten, Opportunitätskosten, Kosten der Verzögerung (Cost of Delay)
  - Investitionen und laufenden Kosten, Run-the-Company (RTC) und Develop-the-Company (DTC)
  - ROI, Break-Even, Abschreibungen und Amortisierung
  - Einmalige und wiederkehrende Aufwände, bekannte und geschätzte Aufwände.

LZ 3-2: Grundbegriffe für Evaluierung/Schätzung kennen und erläutern (R2)

- Begriffe „Schätzung“, „Beobachtung“ und „Messung“ erklären und in der Evaluierung von Problemen und Lösungsansätzen anwenden
- Schätzungen in Intervallen angeben können. Sie kennen verschiedene Ansätze für Intervallschätzungen:
  - Konfidenzintervall
  - Minimum-Maximum-Intervall
  - Worst-/Average-/Best-Case-Schätzung
- Wahrscheinlichkeit der Korrektheit von Schätzungen beurteilen und angeben können (etwa durch Größe von Intervallschätzungen)
- Schätzgegenstände identifizieren und benennen, beispielsweise Arbeitsstunden
- Schätzparameter und Einflussfaktoren auf Schätzungen identifizieren

- Explizite Annahmen, um Schätzparameter in Wertebereichen festlegen zu können.

LZ 3-3: Probleme und Lösungsansätze schätzen können (R1)

- Schätzverfahren auf Probleme und Lösungsansätze innerhalb von IT-Systemen und zugehörigen Prozessen anwenden können

4 Verbesserung langfristig planen

Dauer: 120 min

4.1 Begriffe und Konzepte

- Explizite Darstellung (Dokumentation) von bewerteten Problemen und Lösungsoptionen
- Gruppierung/Clusterung von Lösungen
- Abhängigkeiten von Problemen und Lösungen
- Mögliche m:n Relation von Problemen und Lösungsansätzen
- Synergieeffekte
- Iterativ-inkrementelles Vorgehen
- Entwicklung und Kommunikation langfristiger Lösungsstrategien

4.2 Lernziele

LZ 4-1: Bewertete Probleme und Lösungsansätze explizit darstellen (R1)

Technische oder manuelle Ansätze zur expliziten Darstellung von bewerteten Problemen und Lösungsansätzen kennen und situationsangemessen auswählen – beispielsweise:

- Issue-Tracker
- Tabellen oder
- Datenbanken.

LZ 4-2: Typische methodische Ansätze für Verbesserung kennen und argumentieren (R2)

Typische Ansätze für Verbesserung kennen, beispielsweise:

- Langfristige kontinuierliche Verbesserung
- Verbesserungs-Releases
- Application-Strangulation
- Kapselung (lokalisierbarer) Probleme oder Risiken in Blackboxes
- Extraktion fachlicher Teile, Abtrennung technischer Teile
- Schrittweise Ablösung problembehafteter Systemteile
- Rewrite
- Outsourcing
- Business Process Reengineering.

Ansätze für langfristige Verbesserung an unterschiedliche Stakeholder kommunizieren und argumentieren können.

LZ 4-3: Effekte von „Rewrite“ versus „kontinuierliche Verbesserung“ einschätzen (R2)

In einer gegebenen Situation die Effekte (Risiken, Vorteile) der Ansätze „komplette Neuentwicklung (Rewrite) gegenüber einer „kontinuierlichen Verbesserung“ einschätzen und argumentieren können.

Verstehen, dass mangelnde Kenntnis von Details (wie etwa Anforderungen, Details von Algorithmen und Prozessen, Qualitätsszenarien, Implementierungsdetails, technische Abhängigkeiten, betriebliche Prozesse) häufig die Option „Rewrite“ einfacher erscheinen lässt, als sie bei Berücksichtigung dieser Details wäre.

5 Typische Ansätze für Verbesserung

Dauer: 360 min

5.1 Begriffe und Konzepte

- Strukturelle versus konzeptionelle Verbesserung
- Prozess- und Produktverbesserung
- Verbesserung in Code, Daten, querschnittlichen Konzepten, Prozessen, Infrastruktur, Analysierbarkeit/Monitoring
- Abbau technischer Schulden
- Verbesserungsmaßnahmen für Quellcode
  - Refactoring
  - Reduktion von Komplexität und Kopplung
  - Erhöhung der Lesbarkeit und Verständlichkeit
- Automatisierung von Prozessen zur Senkung von Änderungsrisiken, insbesondere automatisierte Tests.

Anmerkung 1: Die nachfolgenden Lernziele sollen lediglich Anhaltspunkte für typische Verbesserungen geben. In konkreten Systemen oder Situationen werden möglicherweise weitere Ansätze notwendig sein, die dieser Lehrplan nicht antizipieren kann.

Anmerkung 2: Schulungen sollten in diesem Kapitel detaillierte technische Ansätze für mögliche Verbesserung thematisieren oder vorstellen.

LZ 5-1: Möglichkeiten zur Optimierung von Entwicklungsprozessen kennen (R3)

- Dezentralisierung bzw. Zentralisierung von Entwicklungsprozessen
- Einsatz iterativer Prozesse zur Reduktion von Risiken
- Verringerung von Wartezeiten und Puffern zur Beschleunigung von Entwicklungsprozessen
- Identifikation kritischer Bestandteile (Personen, Organisationseinheiten) in Entwicklungsprozessen und deren Entlastung

LZ 5-2: Verbesserungsmaßnahmen im Quellcode kennen und anwenden können (R1)

- Für Technologien/Programmiersprachen typische Refactorings (bedeutungserhaltende Vereinfachungsmassnahmen im Quellcode) kennen und anwenden können. In Schulungen sollten bei Bedarf entsprechende Übungen durchgeführt werden.
- Technologien besser nutzen und bessere Technologie einführen (Technologiewechsel).

- Zusammenhang zwischen technischen Schulden und Refactoring erklären können.
- Maßnahmen und Muster zur Reduktion von Kopplung in Quellcode kennen und anwenden können.
- Maßnahmen und Muster zur Verbesserung der Verständlichkeit von Quellcode kennen und anwenden können, beispielsweise Clean-Code-Prinzipien.

LZ 5-3: Möglichkeiten des automatisierten Tests zur Reduktion von Änderungsrisiken kennen und anwenden können (R2)

- Automatisierte Tests (Unit-, Integrations-, Akzeptanz-, Regressionstests) als Mittel zur Früherkennung von Fehlern in Änderungs- oder Verbesserungsprojekten kennen und selbständig anwenden können (In Schulungen sollten bei Bedarf entsprechende Übungen durchgeführt werden).
- Selbständig identifizieren können, an welchen Stellen/Schnittstellen/Komponenten die Einführung automatisierter Tests für bestimmte Änderungsszenarien angemessen ist.

LZ 5-4: Automatisierung als Mittel zur Reduktion von Änderungsrisiken kennen (R2)

Weitere methodische und technische Mittel zur Automatisierung kennen, mit deren Hilfe systematisch Änderungsrisiken und -aufwände gesenkt werden können, beispielsweise:

- Continuous Integration
- Continuous Delivery
- Model Driven Development/Generative Development.

LZ 5-5: Relevante Architektur-/Entwurfs- oder Implementierungsmuster zur Reduktion von Kopplung innerhalb von Systemen kennen (R1)

- Typische Muster zur Reduktion von Kopplung kennen und anwenden können (beispielsweise Modularisierung/Komponentenbildung, Entkopplung über Interfaces, Dependency-Injection, Kapselung, Adapter, Wrapper, Gateway), zeitliche Entkopplung über Asynchronität).
- Effekte typischer Muster (ggfs. unter Zuhilfenahme entsprechender Werkzeuge) nachvollziehen können.

LZ 5-6: Technologiespezifische Möglichkeiten zur Verbesserung des Laufzeitverhaltens von Systemen kennen (R1)

Technologiespezifische Muster und Praktiken zur Verbesserung von Laufzeiteigenschaften kennen und anwenden können (die detaillierte Auswahl obliegt dem Schulungs-/Trainingsanbieter).

LZ 5-7: Möglichkeiten zur Verbesserung betrieblicher Prozesse kennen (R2)

(Unter Umständen technologiespezifische) Muster und Praktiken zur Verbesserung der Betreibbarkeit kennen (die detaillierte Auswahl obliegt dem Schulungs-/Trainingsanbieter).

LZ 5-8: Möglichkeiten zur Verbesserung der Dokumentation oder Dokumentationsprozesse kennen (R2)

Grundlegende Möglichkeiten zur systematischen Verbesserung von technischer Dokumentation kennen und anwenden können, beispielsweise:

- Einhaltung etablierter Dokumentationsstrukturen (z. B. Templates)
- Gezielte Reduktion von Dokumentationsumfang durch Abstraktion oder inhaltliche Fokussierung
- Top-Down-Kommunikation
- Trennung struktureller (spezifischer) und konzeptioneller (übergreifender) Inhalte
- Modularisierung von Dokumentation

6 Beispiele für Verbesserung

Dauer: 180 min

Anmerkung: Beispiele für Verbesserungsprojekte realer Systeme können durch Trainer und Schulungsanbieter individuell ausgesucht werden.

Aufgrund der bei realen IT-Systemen oftmals geltenden Geheimhaltungs- oder Vertraulichkeitsregelungen stellt der iSAQB e. V. Schulungsanbietern und Trainern frei, Beispiele abstrahiert oder nur auszugsweise in Schulungen zu zeigen.

LZ 6-1: Beispiele für Probleme/Risiken in IT-Systemen kennen und nachvollziehen (R2)

- Teilnehmer sollen an mindestens einem Beispiel Probleme und Risiken eines mittleren/großen IT-Systems erkennen und nachvollziehen können. Dazu sollten in Schulungen mindestens dessen fachliche/funktionale Anforderungen, wesentliche Qualitätsziele, Anwendungs- und Änderungsszenarien, wesentliche Implementierungsstrukturen sowie wesentliche übergreifende Konzepte beschrieben werden.

LZ 6-2: Bewertung von Problemen/Risiken kennen und nachvollziehen (R2)

- Teilnehmer sollen für ein Beispiel die Bewertung (Evaluierung) von Problemen oder Lösungsansätzen nachvollziehen können.

LZ 6-3: Mittel- bis langfristige Planung eines Verbesserungsprojektes kennen (R3)

- Teilnehmer sollen für ein Beispiel die (kurz-, mittel- und/oder langfristige) Planung eines Verbesserungsprojektes kennenlernen und nachvollziehen können.

LZ 6-4: Verbesserungsmaßnahmen eines realen Projektes kennen und nachvollziehen (R2)

- Teilnehmer sollen für ein Beispiel die möglichen oder durchgeführten Verbesserungsmaßnahmen kennenlernen und nachvollziehen können.

7 Quellen und Referenzen zu IMPROVE

A

[aim42.org] Architecture Improvement Method – Open-Source Sammlung von Praktiken und Patterns rund um die Verbesserung von Softwarearchitekturen und -systemen. http://aim42.org sowie http://aim42.github.io

B

[Bass+2012] Bass, L., Clements, P. und Kazman, R.: Software Architecture in Practice. 3rd Edition, Addison-Wesley, 2012

[Bommer+2008] Softwarewartung: Grundlagen, Management und Wartungstechniken. Dpunkt.verlag, 2008

C

[Clements+2001] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures Methods and Case Studies. Addison-Wesley, 2001

F

[Feathers] Working Effectively with Legacy Code. Prentice Hall, 2007

[Fowler+99] Martin Fowler, Kent Beck: Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

L

[Lippert+2007] Martin Lippert, Stefan Roock: Refactoring in Large Software Projects: Performing Complex Restructurings Successfully. Wiley, 2007.

[Lilienthal 2016] Carola Lilienthal: Langlebige Softwarearchitekturen, Technische Schulden analysieren, begrenzen und abbauen. Dpunkt.verlag, 2016

M

[McConnell-2006] Steve McConnell: Software Estimation: Demystifying the Black Art. Microsoft Press, 2006.

[Murer+2010] Stephan Murer, Bruno Bonati: Managed Evolution: A Strategy for Very Large Information Systems. Springer, 2010