Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
IMPROVE

Evolution and Improvement of Software Architectures

Version 1.6.1-EN, 2. Jan 2020

Table of Contents

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2019

The curriculum may only be used subject to the following conditions:

1. You wish to obtain the CPSA Certified Professional for Software Architecture Advanced Level® certificate. For the purpose of obtaining the certificate, it shall be permitted to use these text documents and/or curricula by creating working copies for your own computer. If any other use of documents and/or curricula is intended, for instance for their dissemination to third parties, for advertising etc., please write to info@isaqb.org to enquire whether this is permitted. A separate license agreement would then have to be entered into.
2. If you are a trainer or training provider, it shall be possible for you to use the documents and/or curricula once you have obtained a usage license. Please address any enquiries to info@isaqb.org. License agreements with comprehensive provisions for all aspects exist.
3. If you fall neither into category 1 nor category 2, but would like to use these documents and/or curricula nonetheless, please also contact the iSAQB e. V. by writing to info@isaqb.org. You will then be informed about the possibility of acquiring relevant licenses through existing license agreements, allowing you to obtain your desired usage authorizations.

Important Notice

We stress that, as a matter of principle, this curriculum is protected by copyright. The International Software Architecture Qualification Board e. V. (iSAQB® e. V.) has exclusive entitlement to these copyrights.

The abbreviation "e. V." is part of the iSAQB's official name and stands for "eingetragener Verein" (registered association), which describes its status as a legal entity according to German law. For the purpose of simplicity, iSAQB e. V. shall hereafter be referred to as iSAQB without the use of said abbreviation.

Introduction: General information about the iSAQB Advanced Level

What does an Advanced Level Module convey?

- The iSAQB Advanced Level offers modular training in three competence areas with flexible pathways through the programme. It acknowledges and supports individual strengths and focus points.
- The certification is based on a homework paper. Grading and oral examination will be carried out through an expert designated by iSAQB.

What qualifications do Advanced Level (CPSA-A) graduates gain?

CPSA-A graduates are able to:

- design medium to large IT systems idependently and based on solid methodical foundations
- take technical and operational responsibility in IT systems with medium to high criticality
- design and document measures to achieve quality requirements and support development teams implementing those measures
- manage communication relevant to architecture in medium to large development teams

Requirements for the CPSA-A certification

- successful training and graduation of Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- at least three years industrial, full-time experience in the IT sector; including collaboration on design and development of at least two different IT systems
  - exceptions may be granted (for example: contributions to open source projects)
- participation at iSAQB Advanced Level trainings worth at least 70 credit points from two different areas of competence
  - existing certifications (for example: Sun/Oracle Java architect, Microsoft CSA) may be credited
- passing the CPSA-A certification exam

Basics

What does the module “IMPROVE” convey?

Participants learn to methodically improve software systems and architectures, guided by economic and technical goals. The trainings impart the systematic separation of problem and solution, the elaboration of short-, mid- and long-term solution strategies as well as their alignment with business goals and measures.

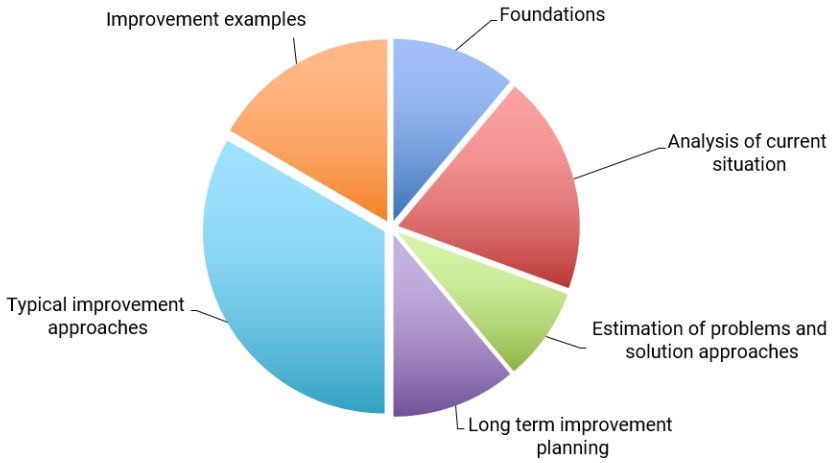
In addition, the IMPROVE curriculum teaches typical approaches of improvement, e. g., restructuring and refactoring, improving analysability, process improvement, improvement of technical infrastructure, improvement of quality attributes, etc.

Curriculum structure and recommended durations

In particular, examples and exercises are left unspecified in this curriculum.

Content	Recommended minimum duration (minutes)
1. Foundations	120
2. Analysis of current situation	210
3. Estimation of problems and solution approaches	90
4. Long term improvement planning	120
5. Typical improvement approaches	360
6. Improvement examples	180
Sum	1080 (18h)

Durations



Duration, didactics, and further details

The durations mentioned below are recommendations. A course for the IMPROVE should last at least 3 days. Provides may vary length, didactics, type and structure of exercises, and structure of the course.

Licensed courses for IMPROVE contribute the following credit points to the Advanced Level graduation:

Methodical Competence:	20 Points
Technical Competence:	10 Points
Communicative Competence:	0 Points

Prerequisites

Participants **should** have the following knowledge and/or experience:

- Hands-on experience in design and development of small to medium size software systems.
- Some hands-on experience in maintenance and evolution of a software system.
- Some hands-on experience with the handling of source code of medium to large scale systems, in particular analysis, assessment, quality check, refactoring, and applying of metrics.

Furthermore, the following will be useful for understanding certain concepts:

- Knowledge and hands-on experience with refactoring (see e. g., [Fowler+99])
- Knowledge or some hands-on experience with review and evaluation of software:
  - Basic knowledge about software metrics, such as metrics to measure coupling, cohesion, dependency and complexity.
  - Basic knowledge of software run time analysis such as profiling, tracing, log analysis and data analysis.
  - Basic knowledge and some experience with the estimation of development efforts.
- Some hands-on experience with requirements engineering as well as contact with various stakeholders in development projects.
- Some hands-on experience with analysis of development processes.

Structure of the curriculum

The sections of the curriculum are described according to the following structure:

- Terms and concepts: essential core terms of the topic
- Teaching/Exercise Time: Specifies the teaching and exercise time that at least has to be spent on this topic or its exercise in an accredited course.
- Learning goals: Describes the contents to be taught, including their key terms and concepts.

Learning goals also outline the knowledge and capabilities to gain in appropriate trainings. They are distinguished into the following categories of relevance:

- (R1) What should the participants be capable of? After completion of the training the participants should be able to autonomously put these contents into practice. These contents will be taught within trainings and should also be deepened through exercises and discussions.
- (R2) What should the participants understand? These contents will be taught within trainings and may be supported by exercises and discussions.

- (R3) What should the participants know? These contents (terms, concepts, methods, practices or similar) may enhance the understanding or motivate the topic. They will be covered in trainings if required but not necessarily in much detail.

If required, the learning goals include references to further literature, standards or other sources.

Further information, terminology, translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Lesson 1: Foundations of Improvement and Evolution of Software Architecture

Lesson duration: 120 min	Exercises: 0 min

1.1. Terms and concepts

- Software architecture: structure, building blocks/components, interfaces, cross-cutting concepts
- Change, evolution, maintenance, improvement of software
- Categories of problems and risks in software (technical debts)
- Core terms related to improvement and change of software
- Technical care.

1.2. Learning goals

LG 1-1: Know and explain reasons for software changes (R1)

- Extending and changing features
- Change in quality requirements and goals
- Changes in technical or business context (e. g., change in external interfaces)
- Bug fixing
- Changes in organisation (e. g., changes of legal conditions or requirements; organisational structure of business units, development or operations)
- Reduce of costs or efforts, especially with respect to:
- Costs or efforts of development
- Costs or efforts of operations
- Costs of bug fixing and consequential damage
- Costs of involved processes
- Opportunity costs
- Intrinsic motivation of stakeholders, especially software developers and architects.
- Update of applied technology such as operating systems, middleware, libraries, frameworks, hardware or similar.

LG 1-2: Know and explain typical categories of problems in software (R1)

- Conceptual weaknesses, e. g., violation of conceptual integrity, redundancy, use of inappropriate technology, incorrect use of technology.
- Structural problems in data or data structures.
- Structural problems in source code, such as:
- Missing or inadequate modularisation,
- Tight coupling,

- High complexity,
- Low cohesion,
- Flaws in programming, e. g., non-idiomatic use of languages or tools.
- Runtime problems in systems, like instability, lack of robustness or reliability, insufficient performance or extensive resource demands, insufficient scalability, insufficient transparency of system processes
- “Legacy” gathered by frequent changes of requirements made under time and/or cost pressure.
- Recognize and name different kinds of technical debts.
- Problems to achieve quality goals or requirements, e. g., insufficient changeability or maintainability, low flexibility, insufficient security or lack of portability.
- Problems in development, operations or other involved processes (e. g., requirements management, test/QA, configuration, deployment, monitoring/alerting).

LG 1-3: Know and explain core terms of evolution and change (R1)

Know core terms of evolution and change:

- Problem (issue),
- Solution approach (opportunity for improvement),
- Costs (of problems, solution approaches, measures),
- Root cause versus symptom,
- Risk.

LG 1-4: Know and explain possible approaches for changes (R2)

For example:

- Ad-hoc improvement, one-off improvement,
- Stepwise improvement (mid- to long-term),
- Improvement by newly developed system or system parts (rewrite),
- Purely structural improvements (refactoring),
- Conceptual/structural improvements (re-architecting, reengineering).

2. Lesson 2: Analyse Current State

Lesson duration: 210 min	Exercises: 0 min

2.1. Terms and concepts

- current state analysis, strength/weaknesses analysis,
- Stakeholder,
- Problem, cause versus symptom,
- Solution approach.

2.2. Learning goals

LG 2-1: Know the basics of the analysis to distinguish “problem” from “solution” and being able to apply it (R1)

- Distinguish “problem analysis” from “problem solution”
- Form m:n relation between problems and solution approaches
- Decomposition of complex problems

LG 2-2: Know typical practices and methods for current state analysis and being able to apply them (R1)

Know typical practices for current state analysis and being able to choose the appropriate method in each situation according to budget, time or the involved stakeholders. This includes approaches such as:

- Stakeholder analysis and interview,
- Contextual analysis,
- Qualitative analysis (e. g., ATAM), in particular, analysis of quality goals,
- Analysis of structural deviations between target and current architecture,
- Quantitative analysis methods and practices, such as:
- Code metrics (size metrics, coupling, complexity, cohesion),
- Organisational metrics, such as costs, time, and countable items. Example: error counts and failure rates, development speed, cost per feature, cost per fixed bug, etc.,
- Runtime metrics, e. g., time and resource demands as well as tools to measure these metrics.
- Data analysis: examination of data structures and contents,
- Documentation analysis,
- Analysis of technical environment (runtime and operations: hardware, operations environment, networks, operating systems involved and infrastructure software)
- Analysis of development processes (requirements engineering, design/implementation, test/QA and handover to operations)
- Analysis of operation processes
- Analysis of further processes and tasks that may influence development problems and system

changes, e. g.,: management, budgeting, support, governance, outsourcing/offshoring, procurement.

LG 2-3: Being able to methodically document identified problems and risks (R1)

- Participant shall be able to initiate an adequate documentation of problems (issues) and risks that have been identified by an improvement- and change process.
- Tools and examples for problem documentation.

LG 2-4: Being able to plan and execute stakeholder analysis and interviews (R1)

- Plan, perform, and document a stakeholder analysis to identify essential people involved, their roles, and intents.
- Know methods to structure and execute stakeholder interviews.
- Being able to create preparatory questionnaires.
- React flexibly to new relevant information obtained during interviews; incorporate these in the analysis.

LG 2-5: Being able to perform a contextual analysis (R1)

- Define and document contextual boundaries: demarcate systems with respect to their technically and logically related neighbours, identify external interfaces.
- Identify connections between external interfaces and stakeholders and use this information for problem analysis.
- Elaborate problems and risks of external interfaces (e. g., via interviews, analysis of known failures, runtime analysis, protocol or log analysis, analysis of organisational dependencies, analysis of quality attributes and/or service levels).

LG 2-6: Being able to perform code- and structural analysis (R1)

- Perform and document (static) analysis of existing source code and its structure. (For this purpose, tools may be used in the training. However, these are not a prerequisite).

LG 2-7: Being able to plan and perform basic runtime analysis (R2)

- Plan and perform (dynamic) analysis of existing systems, e. g., with respect to runtime behaviour, resource utilization, load response. (For this purpose, tools may be used in the training. However, these are not a prerequisite).

3. Lesson 3: Estimate and Assess Problems and Solution Approaches

Lesson duration: 90 min	Exercises: 0 min

3.1. Terms and concepts

- Effort, cost, estimate, observation/measurement, assumptions
- economic dimensions
  - Investment, yield, cost, value
  - Return-on-Invest (ROI)
  - Break-Even,
  - RTC and BTC costs
- Interval estimation, Law of large numbers.

3.2. Learning goals

LG 3-1: Know and explain economic dimensions (R1)

- Explain costs and value as linked concepts.
- Name and classify different kinds of economic dimensions, such as:
- Direct costs, education costs, capital expenditures (direct and indirect cost of investment), operating expenditures, opportunity cost, and cost of delay
- Investment and operating expenses (OPEX), Run-the-Company (RTC) and Develop-the-Company (DTC)
- ROI, break-even, depreciation and amortisation
- One-time and reoccurring costs, known and estimated costs.

LG 3-2: Know and explain basic terms of evaluation and estimation (R2)

- Explain the terms “estimation“, “observation“ and “measurement“ and apply them to the evaluation of problems and solution approaches.
- Be able to estimate efforts in intervals. Know different approaches for interval estimates:
- Confidence interval,
- Minimum/maximum interval,
- Worst-/Average-/Best-Case estimation.
- Be able to assess and communicate the probability of the correctness of the estimation.
- Identify and name items to be estimated, e. g., hours of work.
- Identify parameters and influencing factors of estimations
- explicit assumption to be able to define estimation parameters in value ranges.

LG 3-3: Being able to estimate for problems and solution approaches (R1)

Apply estimation techniques to problems and solution approaches of IT systems and related processes.

4. Lesson 4: Long-Term Planning of Improvements

Lesson duration: 120 min	Exercises: 0 min

4.1. Terms and concepts

- Explicit representation (documentation) of evaluated problems and options for solutions.
- Grouping/clustering of solutions.
- Dependencies between problems and solutions.
- Potential m:n relation between problems and solution approaches.
- Synergies,
- Iterative-incremental approach.
- Development and communication of long-term solution strategies.

4.2. Learning goals

LG 4-1: Explicitly represent evaluated problems and solution approaches (R1)

Know technical or manual approaches to explicitly represent evaluated problems and solution approaches and choose the appropriate for the given situation. Examples:

- Issue tracker,
- Tables,
- Data bases.

LG 4-2: Know and argue typical methods for improvement (R2)

Know typical methods for improvement, e. g.,:

- Long-term continuous improvement,
- Improving releases,
- Application strangling,
- Encapsulation of (localised) problems or risks in black-boxes,
- Extraction of business aspects, separation of technical aspects,
- Incremental replacement of problematic parts of system,
- Rewrite,
- Outsourcing,
- Business process reengineering.

Being able to communicate and argue approaches to long-term improvement to different stakeholders.

LG 4-3: Assess the impact of “rewrite“ versus “continuous improvement“ (R2)

Being able to assess and argue the impact (risks, benefits) of a “complete rewrite” approach in contrast to a “continuous improvement“ approach in each situation.

Understand that lack of details knowledge (e. g., requirements, details of algorithms and processes, quality scenarios, implementation details, technical dependencies, operational processes) often leads to the “rewrite” approach looking simpler as it would be when considering all the details.

5. Lesson 5: Typical Approaches to Improvement

Lesson duration: 360 min	Exercises: 0 min

5.1. Terms and concepts

- Structural vs. conceptual improvement
- Process and product improvement
- Improvement of code, data, cross-cutting concepts, processes, infrastructure, analysability/monitoring
- Reduction of technical debts
- Ways to improve source code
  - Refactoring
  - Reduction of complexity and coupling
  - Improve readability and comprehensibility
- Process automation to lower risk of changes, in particular, automated tests.

Note 1: The following learning goals should only give indications for typical improvements. In concrete systems or scenarios other approaches may be necessary that cannot be anticipated in this curriculum.

Note 2: In this chapter trainings should present detailed technical approaches for possible improvements.

5.2. Learning goals

LG 5-1: Know potential approaches to optimise development processes (R3)

- Decentralisation vs. centralisation of development processes.
- Employment of iterative processes to reduce risks.
- Reduce idle times and buffers to accelerate development processes.
- Identify critical parts (people, organisational units) in development processes, and possible ways to relieve them.

LG 5-2: Know and being able to apply improvement measures for source code (R1)

- Know and being able to apply typical technology/programming language specific refactorings (semantics preserving simplification measures in source code). Trainings should conduct exercises if required.
- Better use of technology or use better technology (change of technology).
- Explain the connection between technical debts and refactoring.
- Know and be able to apply measures and patterns to reduce coupling at source code level.
- Know and be able to apply measures and patterns to make source code more comprehensible, e. g., Clean-Code principles.

LG 5-3: Know and be able to apply possible approaches for automated testing to reduce risks of change (R2)

- Know and being able to autonomously apply automated tests (unit-, integration-, acceptance-, regression-tests) as a means of early detection of defects in change or improvement projects. (Trainings should provide exercises if required.)
- Being able to autonomously identify adequate locations/interfaces/components where the introduction of automated tests will be suitable for specific change scenarios.

LG 5-4: Know automation as a means of reducing risks of changes (R2)

Know further methodical and technical means of automations that may help to reduce risk and required efforts of changes, such as:

- Continuous integration,
- Continuous delivery,
- Model driven development/generative development.

LG 5-5: Know relevant architectural-/design- or implementation patterns to reduce system-internal coupling (R1)

- Know and be able to apply typical patterns to reduce coupling (e. g., modularisation/component building, decoupling via interfaces, dependency injection, encapsulation, adapter, wrapper, gateway, decoupling of control flow with asynchronous invocation).
- Understand the impact of typical patterns (possibly by using appropriate tools).

LG 5-6: Know technology specific options to improve the runtime behaviour of systems (R1)

Know and be able to apply technology specific patterns and practices to improve runtime properties (specific choices are at the training provider's discretion).

LG 5-7: Know options to improve operation processes (R2)

(Possibly technology specific) patterns and practices to improve system operations (specific choices are at the training provider's discretion).

LG 5-8: Know options to improve the documentation or the documentation processes (R2)

Know and be able to apply basic options for systematic improvement of technical documentation, such as:

- Compliance with established document structures (e. g., templates)
- Targeted reduction of documentation volume through abstraction or focussing on essential topics.
- Top-down communication,
- Separation of structural (specific) and conceptual (overarching) contents.
- Modularisation of documentation.

6. Lesson 6: Examples of Improvement

Lesson duration: 180 min	Exercises: 0 min

This section is not examinable.

6.1. Terms and concepts

In every licensed training session, at least one example for IMPROVE must be presented.

Type and structure of the examples presented may depend on the training and participants' interests. The are not prescribed by iSAQB.

6.2. Learning goals

LG 6-1: Know and understand examples of problems/risks in IT systems (R2)

- Participants should be able to identify and understand the problems and risks of at least one example of a mid- or large-size IT system. Therefore, the trainings should at least describe the system’s main functional and non-functional requirements, quality goals, usage and change scenarios, essential implementation structures as well as important cross-cutting concepts.

LG 6-2: Know and understand the evaluation of problems/risks (R2)

- Participants should be able to understand the evaluation of problems or solution approaches in a given example.

LG 6-3: Know medium- to long term planning of an improvement project (R3)

- Participants should know and be able to understand the (short-, mid- and/or longterm) planning of an improvement project in a given example.

LG 6-4: Know and understand improvement measures of a real-life project (R2)

- Participants should know and be able to understand possible or implemented improvement measures in a given example.

References

This section contains references that are cited in the curriculum.

A

[[[aim42+2017]]] Architecture Improvement Method – Open-Source collection of practices and patterns to support software evolution, modernization, maintenance, migration and improvement of software systems. http://aim42.org as well as 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+2007]]] Working Effectively with Legacy Code. Prentice Hall, 2007.

[[[Fowler+1999]]] 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.