

arc⁴²

Gernot Starke
Peter Hruschka

Communicating Software Architectures

lean, effective &
painless documentation

innoQ

Communicating Software Architectures with arc42

Lean, effective & painless documentation

Gernot Starke and Peter Hruschka

This book is for sale at <http://leanpub.com/arc42inpractice>

This version was published on 2016-11-24



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2015 - 2016 Gernot Starke and Peter Hruschka

Contents

Acknowledgments	i
I. Introduction	1
I.1 Basic Principles of arc42	3
I.2 Why This Book	4
I.3 What This Book is NOT	5
I.4 Our Assumptions About You	6
I.5 Quick Navigation	6
I.6 Conventions	8
II. arc42 by Example	9
II.1. Introduction and Goals	10
II.2 Constraints	15
II.3 Context	16
II.4 Solution Strategy	19
II.5 Building Block View	20
II.6 Runtime View	26
II.7 Deployment view	28
II.8 Technical and Crosscutting Concepts	31
II.9 Design Decisions	37
II.10 Quality Scenarios	39
II.11 Risks and technical debt	41
II.12 Glossary	42
It's not the end, is it?	43

Acknowledgments

Many people helped in the last few years with constructive comments and critique, advice and questions around arc42.



Ralf D. Müller is the good soul and tireless committer in the arc42 open-source universe: He answers support questions, maintains the tool-chain that generates arc42 from its AsciiDoc sources and keeps our Github-issues in check. Thank you so much!

We like to thank Martin Dungs, Uwe Friedrichsen, Phillip Ghadir, Mahbouba Gharbi, Franz Hofer, Prof. Arne Koschel, Jürgen Krey, Anton Kronseder, Prof. Bernd Müller, Alex Nachtigall, Axel Noellchen, Robert Reiner, Markus Schärtel, Roland Schimmack, Michael Simons, Boris Stumm, Daniel Takai, Eberhard Wolff, Oliver Wronka and Stefan Zörner for their support during preparation of this book.

Special thanx to Pedro Lafuente Blanco, Stefan Paal, Christopher Schmidt, Silvia Schreier, Per Starke and Oliver Tigges for their constructive and insightful reviews and suggestions.

innoQ¹ supports arc42 by hosting the [arc42 Confluence](http://confluence.arc42.org)² wiki and the arc42.org³ website. Christian Sarazin clears away all the technical debris. Thanx to my skillful, knowledgeable, competent and critical colleagues.

Cover design by [Andreas Steinbrecher](http://andreassteinbrecher.de)⁴.

Parts of translation from the German original were supported by Sven Johann and Per Starke.

Gernot: thanks Uli, Lynn und Per: You are superb, the best family in the universe. My time with you is always too short.

Peter: Special thanks to my wife Monika. You not only survived yet another book project but enriched it by your helpful insights and comments from the non-IT world.

¹<http://innoq.com>

²<http://confluence.arc42.org>

³<http://arc42.org>

⁴<http://andreassteinbrecher.de>

This is just a sample

Dear Readers,

thanx for downloading this sample. It contains chapters I and II of the book. In [section I-5](#) you get an overview of the complete book (and see what topics are missing from this sample).



Many internal references (hyperlinks) will not work in this sample - especially those pointing to sections and tips within chapters III to VII...

I. Introduction

May the force of the proper word and diagram be with you.



This chapter covers the following topics:

- Basic principles of arc42
- Why this book?
- For which purpose can you use arc42?
- What this book is *not*!
- For whom we wrote this book?
- [Quick navigation](#)
- [Table of Tips](#)
- [Table of FAQs](#) (frequently asked questions)

Often software systems are in use for many years - and are constantly maintained and improved. Sometimes the life of software comes to a tragic end: missing maintainability and insufficient documentation are a risky game with uncertain outcome even for small changes. Marginal extension can only be done with massive efforts. The business value of the system fades away.

As a software architect, design your systems in a way that they are maintainable, flexible and understandable, thus thoroughly avoiding their „decay“. To achieve that you should not only fulfill the functional requirements but also concentrate on inner qualities of your system. One of your jobs is to adequately communicate its architecture in both written form and by word of mouth.

In this book we introduce you to arc42, a proven, practical standard for documenting and communicating software architecture – available free of charge! arc42 is based on years of experience and used successfully since 2005 in many companies and organizations in different application domains.

First of all arc42 is a **template for architecture documentation**.

It answers the following two questions in a pragmatic way, but can be tailored to your specific needs:

- *What* should we document/communicate about our architecture?
- *How* should we document/communicate?

Figure I.1 gives you the big picture: It shows a (slightly simplified) overview of the structure of arc42.

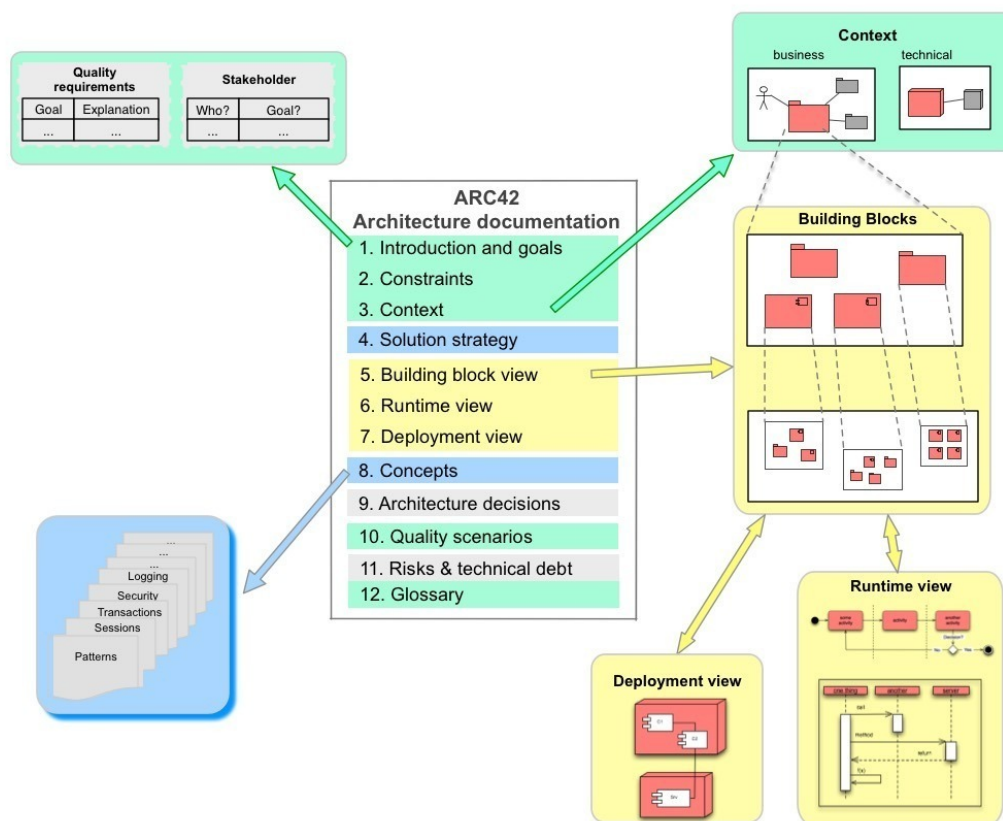


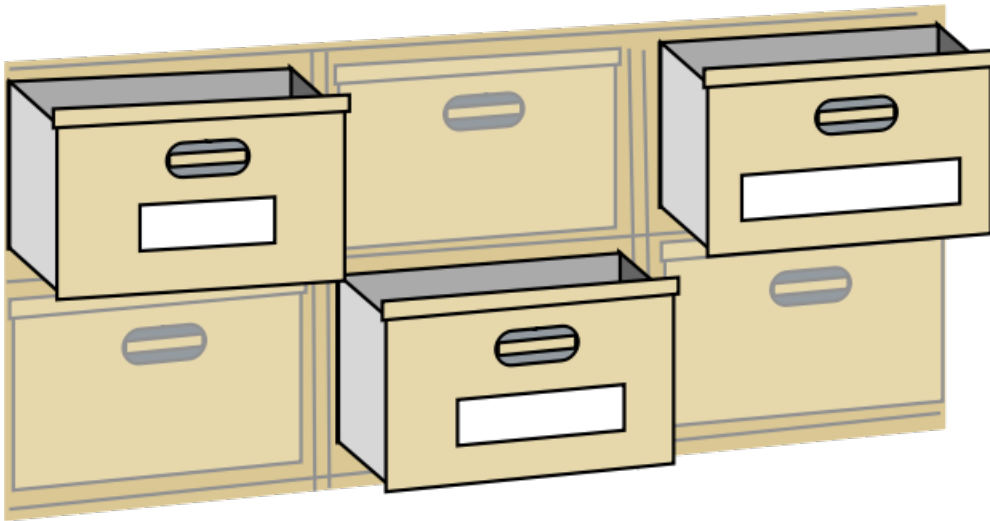
Figure I.1

In case you are impatient and immediately want to see a worked-out example of arc42-documentation, you can jump to [chapter II](#). It will show you how arc42-documentation *feels like*.

For the rest of you (or readers returning from chapter II) we will give you background information that will make your work with arc42 easier.

I.1 Basic Principles of arc42

Clear Structure



arc42 metaphor

Compare arc42 to a cabinet⁵ with drawers: the drawers are clearly marked with labels indicating the content of each drawer. arc42 contains 12 such drawers (a few more than you see in the picture above). The meaning of these arc42 drawers is easy to understand.

Therefore, arc42 offers you a simple and clear structure to document and communicate your (complex!) system. Starting with the goals and requirements for your system and its embedding into its environment you can provide the important stakeholder of your system with adequate information about the architecture.

arc42 is optimized for understandability and adequacy. It naturally guides you to explain any kind of architecture information or decision in an understandable and reproducible context.

Individuals and organizations using arc42 especially like two things about it:

1. the *understandability* of the documentation resulting from its standardized structure (the *drawers*) and
2. the *manageable effort* to create such documentation. We call it “*painless documentation*”.

Independent of process models

You can work on the drawers of the arc42 cabinet in any order – whatever seems useful and adequate under your constraints. When you start a development from scratch you will probably start with requirements and goals; when you maintain or extend existing systems you may dive into the details of the building blocks immediately.

⁵Cabinet image from [Openclipart](#)

With arc42 you can start and stop working on your architecture documentation any time. The fixed structure of your *cabinet* enables you to continue any time. The only prerequisite is an agreed understanding among all team members about the meaning of the arc42-drawers.

Therefore arc42 is completely *process agnostic*.

Architecture documentation with little effort

The arc42 template is a framework for creativity and architecture work, a basis for discussion and feedback among your stakeholders, support for familiarization of rookies and much more.

arc42 is available under a liberal open source license. Therefore you can use it free of charge, even in commercial environment. Working with the arc42 template does not require additional effort for you and your team:

- You only describe things that your stakeholders really have to know.
- You explain facts and issues that are necessary to understand the system or individual design decisions.
- You only keep track of important architecture decision that you had to make anyhow.

arc42 helps you to find the most adequate drawer where facts, features and decisions are kept, so that all stakeholders can easily find them again.

Risk: Template Zombies⁶

We want to warn you about one risk when using arc42 template: users could interpret the template as a form and could be tempted to fill every field. We seriously dislike terms like “fill in” and “form”.

arc42 is intended to be a lightweight tool that can easily be adapted to your specific need. It is not nearly a “fill-in all fields” form.

Our deep aversion against forms and our fear of misuse of arc42 has motivated us to describe a series of basic tips (in [chapter III](#)). You should especially obey [tip III-2](#) (economy) and [tip III-3](#) (adequacy).



I.2 Why This Book

Organizations apply arc42 since 2005 to document software architectures. Since that time we (the authors) helped to introduce arc42, restructure existing documentations according to the clear structure of arc42 or develop and document new systems with arc42.

The downloadable template contains short hints and tips for each parts (the *drawers*). The overall structure is easy to understand; there are no hurdles to use the template.

⁶The term „template zombies“ has been coined by Tom DeMarco, Peter Hruschka et al. in the award-winning book „Adrenalin Junkies and Templates Zombies“ (Dorset House 2007)

Despite this simple and clear structure we have been confronted with a lot of practical every-day questions. We have answered those questions in this book – in the style of the *missing manual*⁷.



What this book explains...

I.3 What This Book is NOT

In this book we focus on effective and efficient use of arc42 to document and communicate software architectures. We explicitly excluded a lot of other topics or disciplines.

This book is *no* introduction to:

- **Software architecture and design.** We assume that you have basic methodology know-how about software design and development. You are able to distinguish problem (requirements) from solutions (architecture, implementation). You know principles like separation of concern, information hiding, loose coupling, strong cohesion, simplicity, high consistency, and you separate domain and technical aspects in your implementation. You have learned about different views of your software architecture and cross-cutting concepts. We collected a few resources in the [appendix](#).
- **Your favorite technology:** Sometimes people confuse specific technologies or frameworks with software architecture. We won't introduce any implementation technologies here, although you need to know about these when you design and implement systems.
- **Architecture and Design Patterns:** Productive software developers and architects immensely benefit by reusing well-proven solution approaches also known as *patterns*. You can find patterns (== proven solutions) for various problem domains, described in excellent printed and online sources. Fire up your favorite search engine...
- **Modeling:** You should be able to use models to abstract static and dynamic aspects of your system. We assume that you know how static models (building block, components, modules and their relationships) relate to the source code. And you also know basics about dynamic models (runtime models, process models)

⁷The *missing manuals*® is a series of book from O'Reilly publishing with the nice subtitle „The book that should have been in the box“.

- **UML (Unified Modeling Language):** We will explain in many of the chapters of this book how to use UML pragmatically and effectively. But we assume that you have basic knowledge about class diagrams, component diagrams, sequence and activity diagrams as well as deployment diagrams. More information can be found in [Booch+05] or [Pilone-05].
- **Requirements Engineering and Business Analysis:** software architects have to understand and potentially clarify requirements for a system. They may have to elicit, document and manage requirements – i.e. do work that ideally would have been done by requirements analysts. We assume that you know about functional requirements and quality requirements and are able to discuss them with your stakeholders. A good source for more information is [Robertson-12].

Although our tips mainly refer to effective usage of arc42, many of them will help you and your team to design and implement better systems.

I.4 Our Assumptions About You

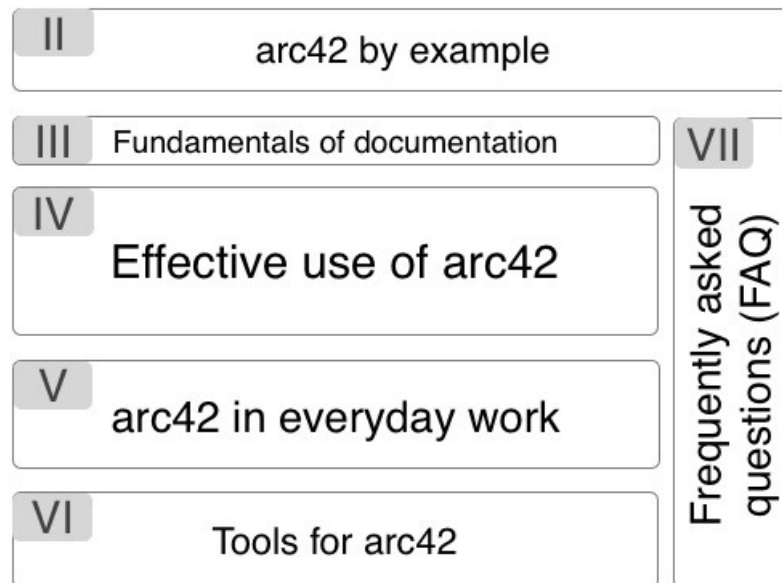
When writing this book, we (the authors) had several (potentially silly) assumptions about you (the readers) in mind:

- First of all, you have **loads of work to do**. Therefore, you only want to read those parts of the book that you consider to be relevant for your work. For this purpose we have written the navigation guide in [section I.5](#).
- You are an **experienced software architect or developer**. Therefore, you are aware of adequate documentation. You have practical experience with the development of architectures and software systems and therefore you are aware what adequate documentation should be.
- You are developing or maintaining mid-size to large, sometimes complex software systems.
- You work under **timing constraints** and only want to read the relevant parts of the book. Therefore, we offer you a navigation guide in [section I.5](#).
- We assume that sometimes in your professional life you have **suffered from missing or excessive documentation**.
- You want to **communicate or document information about the architecture**, the structure and the implementation of your system.
- You do not want to waste time; you want to keep the effort on an adequate level: for some systems you will need detailed information; for other systems it is sufficient to concentrate on core topics.
- Maybe you are already aware of arc42 and want to know how to apply the template more easily, more effective or with more pragmatism.

I.5 Quick Navigation

Since you are very busy in your everyday life you might be interested in identifying the parts of this book that are most relevant for your concrete problems.

The following diagram gives an overview of this book. We have numbered the main chapters with roman numerals (I, II, III) so that you can easily distinguish book chapters from the section numbers in the arc42 template - for which we use the normal Arabic numerals 1 to 12.



Chapter II demonstrates the use of the arc42 template via a small open-source system. In this example you will see how a concrete architecture documentation could look like. For each of the 12 sections of the template you will also find a short motivation explaining why this section is in the template and what should be captured. You can read [chapter II](#) independently from the rest of the book.

Chapter III explains some basis rules of adequate architecture documentation, especially our pledge for “systematic thriftiness”.

Chapter IV contains tons of practical tips for each section of arc42 (i.e. for each drawer – to pick up the metaphor from the introduction). We briefly motivate each arc42 section. Therefore, there is deliberate (small) redundancy with Chapter II. Here we give answers to the questions that we received from our users and seminar participants. This is the most extensive part of the book.

Chapter V explains how to use arc42 in everyday life. You will find hints for creating new systems or amending existing systems, for agile projects and for very large projects with many subsystems.

Chapter VI introduces tools and tool categories you can use to bring arc42 to life.

Chapter VII answers frequently asked questions in various categories (sometimes referring to the tips of Chapter III to VI). We maintain and update this FAQ chapter [online](#)⁸.

⁸<http://faq.arc42.org>

I.6 Conventions



To direct your attention to specific tips or statements we use graphical icons to categorize tips or advice:



Important:

Despite thriftiness and agility there are some informations about your system that you should always document; i.e. quality goals of your architecture.



Economical:

You are looking for opportunities to shorten or streamline you documentation pragmatically. You want to reduce efforts without losing content or value. You are working in an agile environment and want to have lightweight documentation – based on the motto: *travel light*.



Rigorous:

You are working in a more formal environment, e.g. developing very large or critical systems with hard quality requirements. Your stakeholders require thoroughness, accuracy and attention to detail. Maybe your systems and there documentation have to be audited.

Tip I-1: Our numbering scheme



In this book we offer more than 200 different tips around arc42. The numbering scheme aligns them with the corresponding book chapters: the roman prefixes (III, IV, V etc.) refers to the book chapter, the arabic number sequentially numbers tips within chapters.

II. arc42 by Example



This chapter shows a working example of arc2:

In this chapter we show how arc42 can be used to document a small open source system.

Convention for this example

At the beginning of each section you find short explanations, formatted in shaded boxes like this.

Chapter IV contains practical tips for each part of arc42.

The system documented here is a small open source system hosted on [Github](#)⁹.

Tip II-1: Improve your arc42 skills by reading additional examples

Luckily there are additional examples for arc42 architecture documentation available from several different authors.

- There's a Leanpub book, [arc42 by Example](#)¹⁰, that contains several examples.
- [Managing bike tours](#)¹¹, by Michael Simons: Java based app, productive since several years.
- [Chess engine](#)¹², by Stefan Zörner (in German).
- Customer relationship management (CRM).
- [VENOM](#)¹³, a large scale (artificial) e-commerce system. Contains fragments from various *real* systems, combined into a single *legacy tragedy*.

If you like YOUR example to be listed here, just drop us an [email](#)¹⁴.

‘nough preamble. Let's get started...

⁹<https://github.com/aim42/htmlSanityCheck>

¹⁰<http://leanpub.com/arc42byexample>

¹¹<http://biking.michael-simons.eu/docs/index.html>

¹²<http://www.dokchess.de/dokchess/arc42/>

¹³<https://github.com/aim42/venom-example>

¹⁴<mailto:info@arc42.de?subject=arc42%20example>

II.1. Introduction and Goals

Content and Motivation

This section shows the driving forces for architecturally relevant decisions, the important use-cases or features, summarized in a few sentences. If possible, refer to existing requirements documentation.

The main goal of this section is enabling your stakeholders to understand the solution, which is detailed in arc42-sections 3 to 12.

HtmlSC supports authors creating digital formats by checking hyperlinks, images and similar resources.

1.1 Requirements Overview


Content and Motivation

You like to briefly explain important goals and requirements, use-cases or features of the system. If available, refer to existing requirements documentation.

Most important: Readers can understand the central tasks of the system, before they encounter the architecture of the system (starting with arc42-section 3).

The overall goal of HtmlSC is to create neat and clear reports, showing errors within HTML files. Below you find a sample report.

HTML Sanity Check Results

1 page	837 checks	29 issues	0.661sec duration	96% successful	
-----------	---------------	--------------	----------------------	--------------------------	--

Results by Page

Page	Checks	Findings	Success rate
index.html	837	29	96%

Results for index.html

location : /Users/gstarke/projects/aim42/aim42-guide/build/docs/index.html

320.61 kByte	837 checks	29 issues	96% successful
-----------------	---------------	--------------	--------------------------

Missing Local Images Check

30 img src attributes checked, 0 missing image files found.

Duplicate Definition of Id Check

400 id checked, 2 duplicate id found.

- id "Impact-Analysis" has 2 definitions.
- id "Measure" has 2 definitions.

Broken Internal Links Check

370 href checked, 22 missing id found.

- link target "invariant" missing (reference count 1)
- link target "technical-debt" missing (reference count 2)
- link target "Alarm" missing (reference count 1)
- link target "improvement-backlog" missing (reference count 4)
- link target "Architecture-Documentation" missing (reference count 1)
- link target "Hierarchical-Quality-Model" missing (reference count 1)
- link target "Stakeholder-Interviews" missing (reference count 3)
- link target "Runtime-Artifact-Analysis" missing (reference count 5)
- link target "Practices" missing (reference count 1)
- link target "systematic-decisions" missing (reference count 1)
- link target "evaluation" missing (reference count 1)
- link target "Monkey-Patching" missing (reference count 1)
- link target "Deprecate-Obsolete-Parts" missing (reference count 1)
- link target "Systematic-Decisions" missing (reference count 1)
- link target "Profiling" missing (reference count 2)
- link target "Aspect-Oriented-Programming" missing (reference count 1)
- link target "categories" missing (reference count 1)
- link target "collect-opportunities-for-improvement" missing (reference count 3)
- link target "Performance-Analysis" missing (reference count 2)
- link target "Clemens-ATAM" missing (reference count 1)
- link target "meta-programming" missing (reference count 1)
- link target "operations" missing (reference count 1)

Missing Local Resources Check

3 anchor tag href attribute checked, 0 missing local resources found.

Missing alt-attribute declaration in image tags

34 image tags checked, 5 missing alt attributes found.

- image "images/analyze-patterns-overview.png" is missing alt-attribute
- image "images/analyze-patterns-conceptmap.png" is missing alt-attribute
- image "images/evaluate-patterns-conceptmap.png" is missing alt-attribute
- image "images/crosscutting-patterns-overview.png" is missing alt-attribute
- image "images/crosscutting-patterns-complete.png" is missing alt-attribute

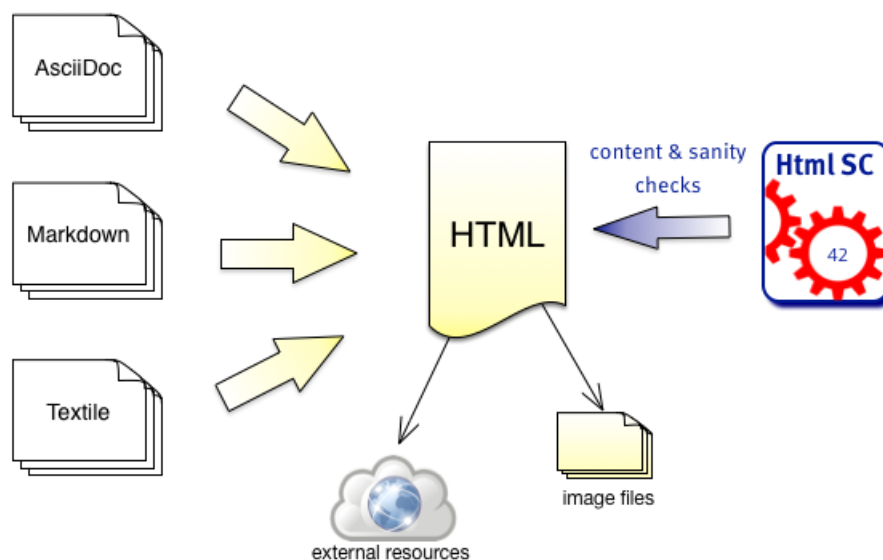
HtmlSanityCheck (HtmlSC) checks HTML for semantic errors, like broken links and missing images.

It has been created to support authors who create HTML as output format.

1. Authors write in formats like [AsciiDoc](http://asciidoctor.org/docs/what-is-asciidoc/)¹⁵, [Markdown](http://www.daringfireball.net/projects/markdown/syntax)¹⁶ or other formats, which are transformed to HTML by the corresponding generators.
2. HtmlSC checks the generated HTML for broken links, missing images and other semantic issues.
3. HtmlSC creates a test report, similar to the well-known unit test report.

¹⁵<http://asciidoctor.org/docs/what-is-asciidoc/>

¹⁶<http://www.daringfireball.net/projects/markdown/syntax>



HtmlSC goal: Semantic checking of HTML pages

Basic Usage

1. A user configures the location (directory and filename) of an HTML file, and the corresponding images directory.
2. HtmlSC performs various checks on the HTML and
3. reports its results either on the console or as HTML report.

HtmlSC can run from the command line or as Gradle plugin.

Basic Requirements

ID	Requirement	Explanation
G-1	Read HTML files	HtmlSC shall read one or several (configurable) HTML files as input.
G-2	Gradle Plugin usage	HtmlSC can be run/used as Gradle plugin
G-3	Command line usage	HtmlSC can be run from the command line (shell)
G-4	Open source license	All required dependencies/libraries shall be compatible with a CreativeCommons license.
G-5	Public repositories	HtmlSC shall be available via public repos, like the Gradle plugin portal.
G-6	Multiple input files	Configurable for a set of files, processed in a single <i>run</i> , HtmlSC produces a joint report.
G-7	Suggestions	When HtmlSC detects errors, it shall identify suggestions or alternatives that would <i>repair</i> the error

Required Functions

HtmlSC shall provide the following checks in HTML files:

ID	Requirement	Explanation
C-1	Missing images	Check all image tags if the referenced image files exist.
C-2	Broken internal links	Check all internal links from anchor-tags (<code>href="#XYZ"</code>) if the link targets "XYZ" are defined.
C-3	Missing local resources	Check if referenced files (e.g. css, js, pdf) are missing.
C-4	Duplicate link targets	Check all link targets (<code>... id="XYZ"</code>) if the id's ("XYZ") are unique.
C-5	Malformed links	Check all links for syntactical correctness.
C-6	Unused images	Check for files in image-directories that are not referenced in any HTML files in this <i>run</i> .
C-7	Illegal link targets	Check for malformed or illegal anchors (link targets).
C-8	Broken external links	Check external links for both syntax and availability.
C-9	Broken ImageMaps	Though ImageMaps are a rarely used HTML construct, HtmlSC shall identify the most common errors in their usage.

Reporting and Output Requirements

ID	Requirement	Explanation
R-1	Various output formats	Checking output in plain text and HTML
R-2	Output to stdout	HtmlSC can output results on stdout (the console)
R-3	Configurable output directories	HtmlSC can store results in file in configurable output directories.

1.2 Quality Goals

Content and Motivation

You want to understand the quality goals (aka architecture goals), so you can align architecture and design decisions with these goals.

These (usually *long term*) quality goals diverge from the (usually *short term*) goals of development projects. Mind the difference! See also [arc42-section 10](#).

Priority	Quality-Goal	Scenario
1	Correctness	Every broken internal link (cross reference) is found.
1	Correctness	Every missing local image is found.
2	Flexibility	Multiple checking algorithms, report formats and clients. At least Gradle, command-line and a graphical client have to be supported.
2	Safety	Content of the files to be checked is <i>never</i> altered.
2	Correctness	Correctness of every checker is automatically tested for positive AND negative cases.
2	Correctness	Every reporting format is tested: Reports must exactly reflect checking results.
3	Performance	Check of 100kB html file performed under 10 secs (excluding Gradle startup)

1.3 Stakeholders

Content and Motivation

You want an overview of persons, roles or organizations that affect, are affected by or can contribute to the system and its architecture. Make the concrete expectations of these stakeholders with respect to the architecture and its documentation explicit. Collect these in a simple table.

Remark: For our simple HtmlSC example we have an extremely limited number of stakeholders, in real-life you will most likely have many more stakeholders!

Role	Description	Goal, Intention
Documentation author	writes documentation with HTML output	wants to check that the resulting document contains good links, image references.
arc42 user	uses the arc42 template for architecture documentation	wants a small but practical example of <i>how to apply arc42</i> .
software developer		wants an example of pragmatic architecture documentation

Additional information

- [Section IV-1](#) contains additional tips regarding requirements in general.
- In reality, the *quality requirements* are pretty often neglected and/or remain *implicit*.
- A more complete overview of quality goals can be given in [section-iv-10](#) on *quality scenarios*

II.2 Constraints

Content and Motivation

You want to know the constraints that restrict your freedom of design decisions or the development process. Such constraints are often imposed by organizations across several IT systems.

HtmlSC shall be:

- platform-independent and should run on the major operating systems (Windows(TM), Linux, and Mac-OS(TM))
- implemented in Java or Groovy
- integrated with the Gradle build tool
- runnable from the command line
- developed under a liberal open-source license

Additional information

- [Section IV-2](#) contains additional tips regarding constraints.

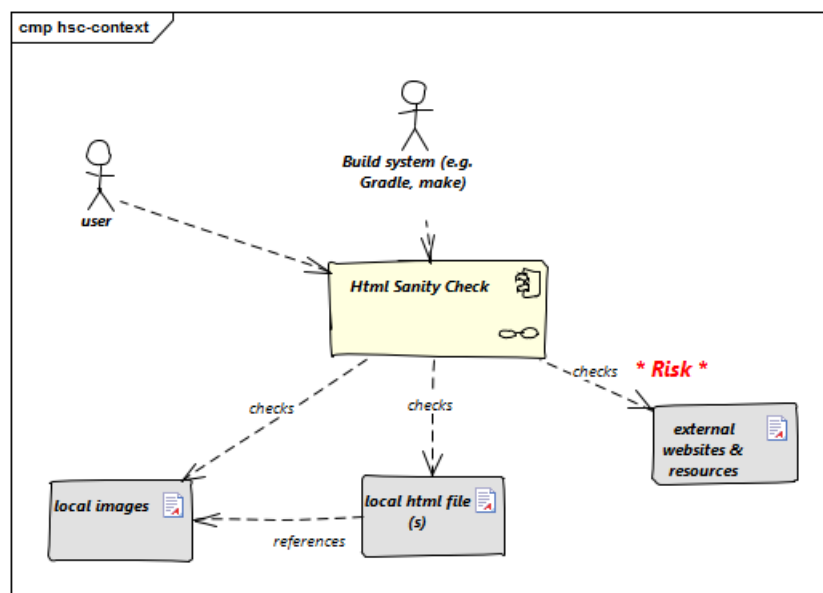
II.3 Context

Content and Motivation

You want to know the boundaries and scope of the system to distinguish it from neighboring systems. The context identifies the systems relevant external interfaces.

3.1 Business Context

You want to identify all neighboring systems and the different kinds of (business) data or events that are exchanged between your system and its neighbors.



Business context

Neighbor	Description
user	documents software with toolchain that generates html. Wants to ensure that links within this HTML are valid.
build system	mostly Gradle ¹⁷
local HTML files	HtmlSC reads and parses local HTML files and performs sanity checks within those.
local image files	HtmlSC checks if linked images exist as (local) files.

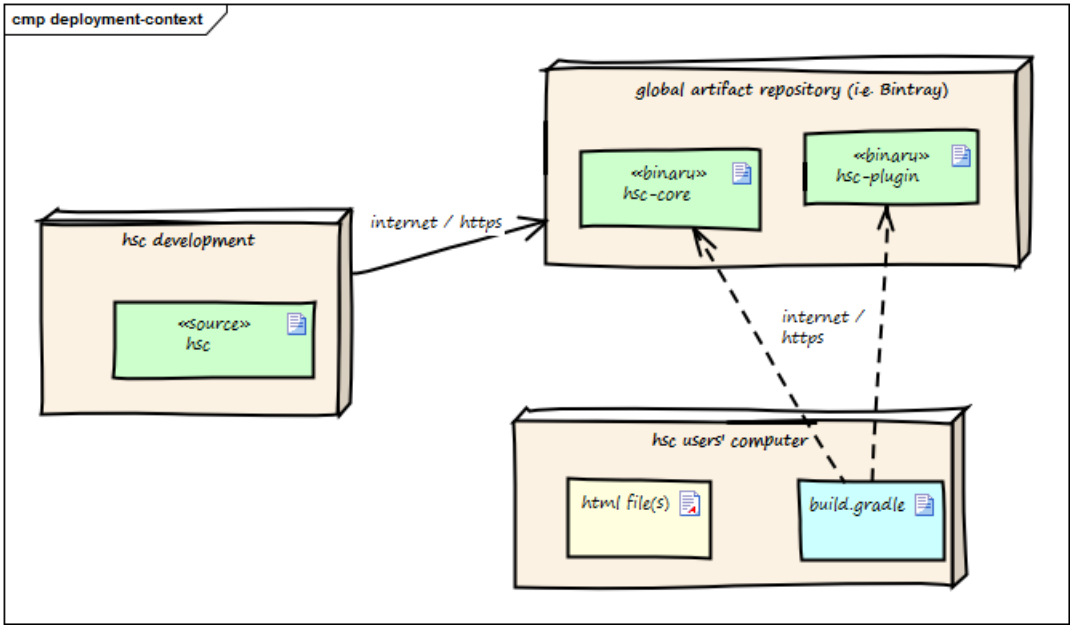
¹⁷<http://gradle.org>

Neighbor	Description
external web resources	HtmlSC can be configured to optionally check for the existence of external web resources. Due to the nature of web systems, this check might need significant time and might yield invalid results due to network and latency issues.

3.2 Deployment Context

You like to know about the technical or physical infrastucture of your system, together with physical channels or protocols.

The following diagram shows the participating computers (nodes) with their technical connections plus the major artifacts of HtmlSC, the hsc-plugin-binary.



Deployment context

Node / Artifact	Description
hsc-development	where development of HtmlSC takes place
hsc-plugin-binary	compiled and packaged version of HtmlSC including required dependencies.
artifact repository ¹⁸	global public <i>cloud</i> repository for binary artifacts, similar to MavenCentral ¹⁹ . HtmlSC binaries are uploaded to this server.
hsc user computer	where arbitrary documentation takes place with html as output formats.

¹⁸<https://bintray.com/bintray/jcenter>[Bintray]
¹⁹<http://search.maven.org/mavenCentral>

Node / Artifact	Description
build.gradle	Gradle build script configuring (among other things) the HtmlSC plugin to perform the HTML checking.

For details see the [deployment-view](#).

Additional information

- [Section IV-3](#) contains additional tips regarding the business and/or technical context.
- Details of the external interfaces shown in the context might be elaborated in the building block view. See [section IV-5](#)

II.4 Solution Strategy

Content and Motivation

You need a brief summary and explanation of the fundamental solution ideas and strategies. These key ideas should be familiar to everyone involved in development and architecture.

Briefly explain how you achieve the most important quality requirements.

1. Implement HtmlSC mostly in the Groovy programming language and partially in Java with minimal external dependencies.
2. We wrap this implementation into a Gradle plugin, so it can be used within automated builds. Details are given in the [Gradle userguide](https://docs.gradle.org/current/userguide/userguide.html)²⁰.
3. Apply the *template-method-pattern*²¹ to enable:
 - multiple checking algorithms. See the [concept for checking algorithms](#),
 - both HTML (file) and text (console) output. See the [reporting-concept](#).

Additional information

- [Section IV-4](#) contains additional tips regarding the solution strategy.
- Keep this part of your documentation *very* short, move extensive explanations, examples etc. to arc42-section 8 (concepts).

²⁰<https://docs.gradle.org/current/userguide/userguide.html>

²¹http://sourcemaking.com/design_patterns/template_method/

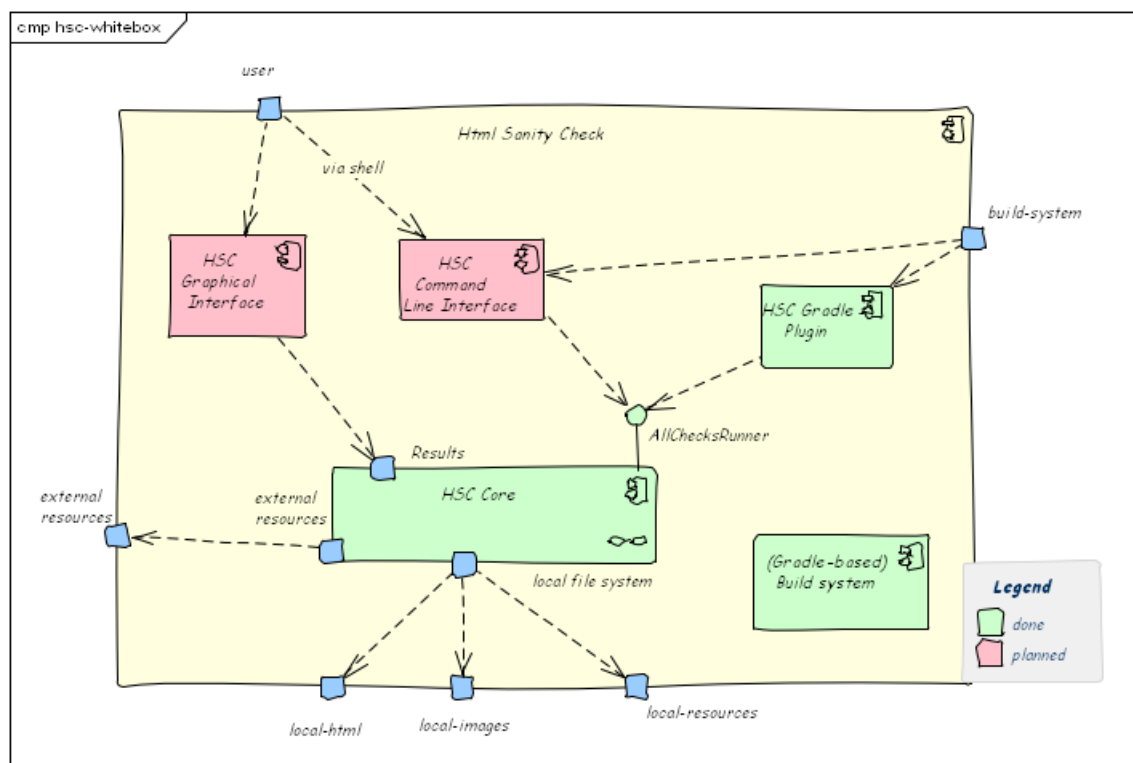
II.5 Building Block View

Content and Motivation

The building block view explains the static decomposition of the system into building blocks (modules, components, subsystems, packages...) and their relationships. It shows the overall structure of the source code.

This view is organized in a top-down hierarchy.

5.1 Whitebox HtmlSanityChecker



Whitebox (HtmlSC)

Rationale: We used *functional decomposition* to separate responsibilities:

- HSC Core shall encapsulate checking logic and HTML parsing/processing.
- HSC Gradle Plugin encapsulates all Gradle specific stuff
- Various kinds of UI (console, graphical) are handled by separate components.

Contained Blackboxes:

Building block	Description
HSC_Core	HTML parsing and sanity checking, file handling
HSC Gradle Plugin	integrates the Gradle build tool with HtmlSC, enabling arbitrary gradle builds to use HtmlSC.
HSC Command Line Interface	(not documented)
HSC Graphical Interface	(planned, not implemented)
(Gradle based) Build System	builds the output artifacts (plugin, jar)

5.1.1 HSC Core (Blackbox)

Intent/Responsibility: HSC_Core contains the core functions to perform the various sanity checks. It parses the html file into a DOM-like in-memory representation, which is then used to perform the actual checks.

Interfaces:

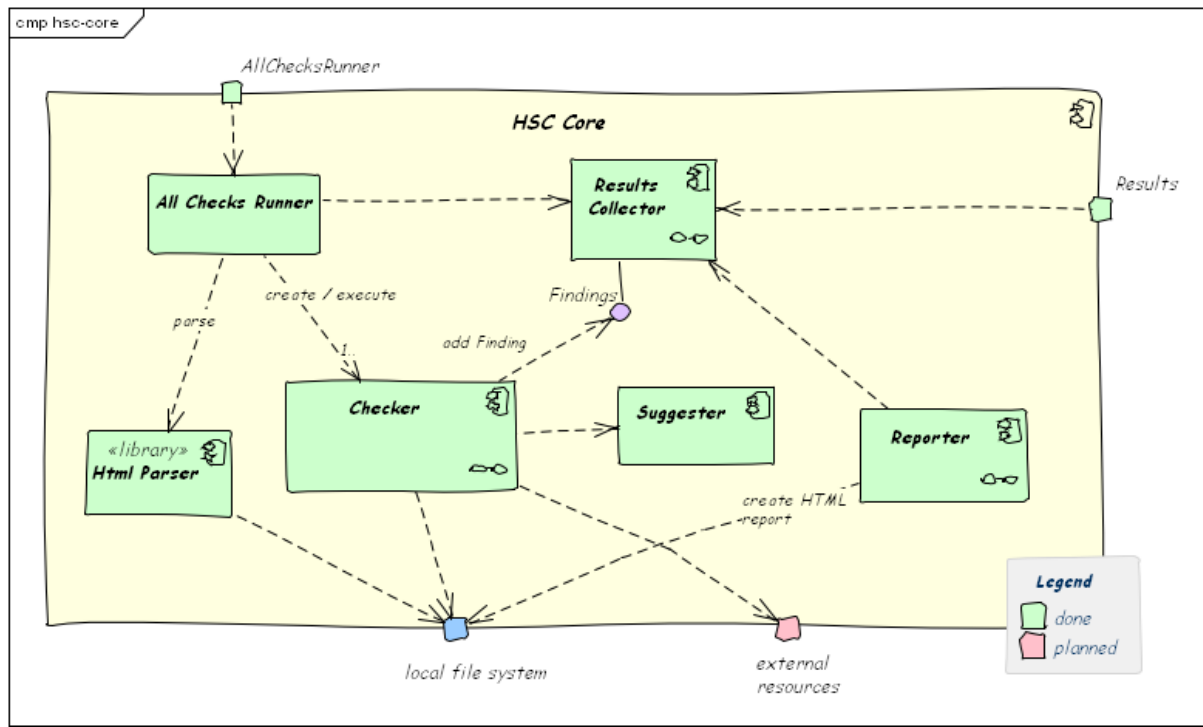
Interface (From-To)	Description
Command Line Interface -> Checker	Uses the AllChecksRunner class.
Gradle Plugin -> Checker	Exposes HtmlSC via a standard Gradle plugin, as described in the Gradle user guide ²² .

Details are described in the [HSC-Core Whitebox](#).

²²<https://docs.gradle.org/current/userguide/userguide.html>

5.2 Building Blocks - Level 2

5.2.1 HSC-Core (Whitebox)



HSC-Core (Whitebox)

Rationale: This structure follows a strictly functional decomposition:

- parsing and handling HTML input,
- checking,
- creating suggestions and
- collecting checking results

Contained Blackboxes:

Building block	Description
Checker	Abstract class, used in form of the template-pattern. Shall be subclassed for all checking algorithms.
AllChecksRunner	Facade to the different Checker instances. Provides a (parameter-driven) command-line interface.
ResultsCollector	Collects all checking results.
Reporter	Reports checking results to either console or file.
HtmlParser	Encapsulates HTML parsing, provides methods to search within the (parsed) DOM tree. We use the open source JSoup ²³ , see the corresponding design decision .

²³<http://jsoup.org/>

Building block	Description
Suggester	In case of checking issues, suggests alternatives (<i>did you mean xyz?</i>). Suggestions are included in results.

Source Files:

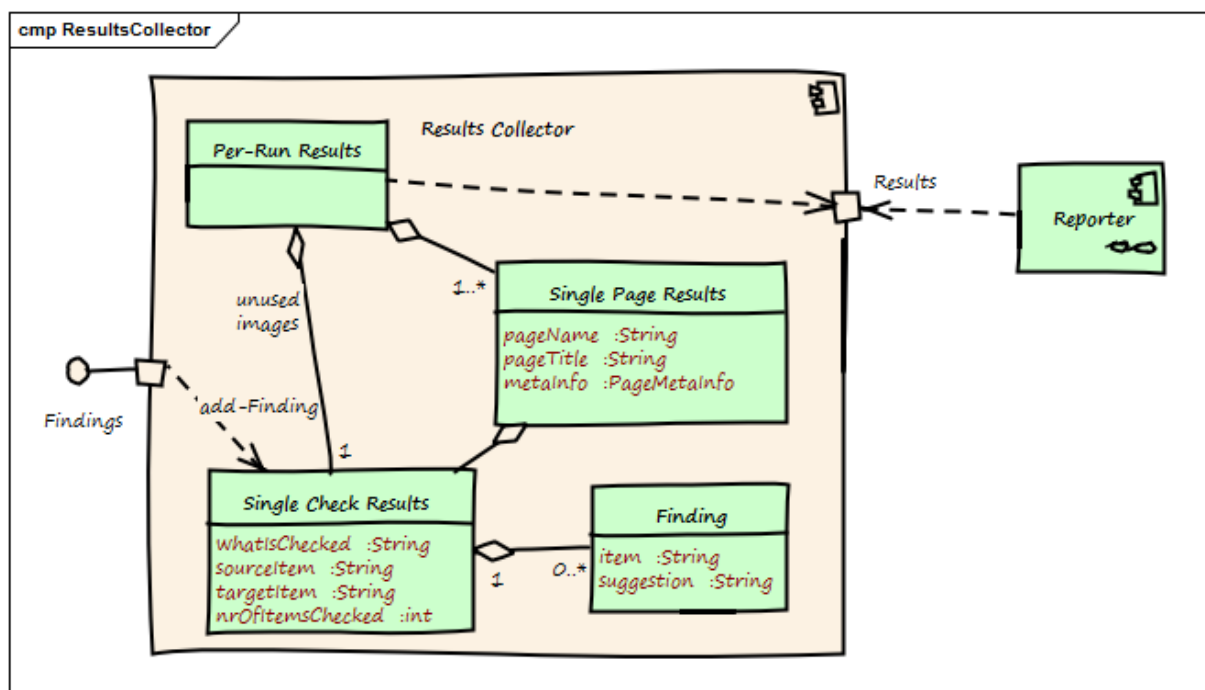
- `org.aim42.htmlsanitycheck.AllChecksRunner`
- `org.aim42.htmlsanitycheck.HtmlSanityCheckGradlePlugin`

5.2.1.1 Checker and <xyz>Checker Subclasses

The abstract Checker provides a uniform interface (`public void check()`) to different checking algorithms. It is based upon the [concept of extensible checking algorithms](#).

5.3 Building Blocks - Level 3

5.3.1 ResultsCollector (Whitebox)



Results Collector (Whitebox)

Rationale: This structure follows the hierarchy of checks, managing results for:

1. a number of pages/documents
2. a single page, each containing many
3. single checks within a page

Contained Blackboxes:

Building block	Description
Per-Run Results	Results for potentially many HTML pages/documents.
SinglePageResults	Results for a single HTML page
SingleCheckResults	Results for a single type of check (e.g. missing-images check or broken-internal-link check)
Finding	A single finding, (e.g. “image ‘logo.png’ missing”). Can contain suggestions.

5.3.2 Interface Results

The `Result` interface is used by all clients (especially `Reporter` subclasses, graphical and command-line clients) to access checking results. It consists of three distinct methods for:

1. overall `RunResults`,
2. single-page results (`PageResults`) and
3. single-check results (`SingleCheckResults`).

See the interface definitions below - taken from the Groovy source code:

Interface RunResults

```
package org.aim42.htmlsanitycheck.collect

public interface RunResults {
    // returns results for all pages which have been checked
    public ArrayList<SinglePageResults> getResultsForAllPages()

    // how many pages were checked in this run?
    public int nrOfPagesChecked()

    // how many checks were performed in all?
    public int nrOfChecksPerformedOnAllPages()

    // how many findings (errors and issues) were found in all?
    public int nrOfFindingsOnAllPages()

    // how long took checking (in milliseconds)?
    public Long checkingTookHowManyMillis()
}
```

Interface CheckResults

```
package org.aim42.htmlsanitycheck.collect

interface CheckResults {
    // description of what is checked
    // (e.g. "Missing Images Checker" or "Broken Cross-References Checker"
    public String description()

    // all findings (aka problems) found during this check
    public ArrayList<Finding> getFindings()
}
```

5.3.3 Suggester (Whitebox)

For a give input (*target*), Suggester searches within a set of possible values (*options*) to find the *n* most similar values. For example:

- Target = “McDown”
- Options = {“McUp”, “McDon”, “Mickey”}
- The resulting suggestion would be “McDon”, because it has the greatest similarity to the target “McDown”.

Suggester is used in the following cases:

- Broken image links: Compares the name of the missing image with all available image file names to find the closest match.
- Missing cross references (broken internal links): Compares the broken link with all available link targets (anchors).

Additional information

- [Section IV-4](#) contains additional tips regarding the building block view.

II.6 Runtime View

Content and Motivation

The runtime view shows behavior, interactions and runtime dependencies of the building blocks in form of concrete scenarios.

It helps you to understand *how* the building blocks of your systems fulfill their respective tasks at runtime, and *how* they communicate/interact with each other at runtime.

II.6.1 Execute all checks

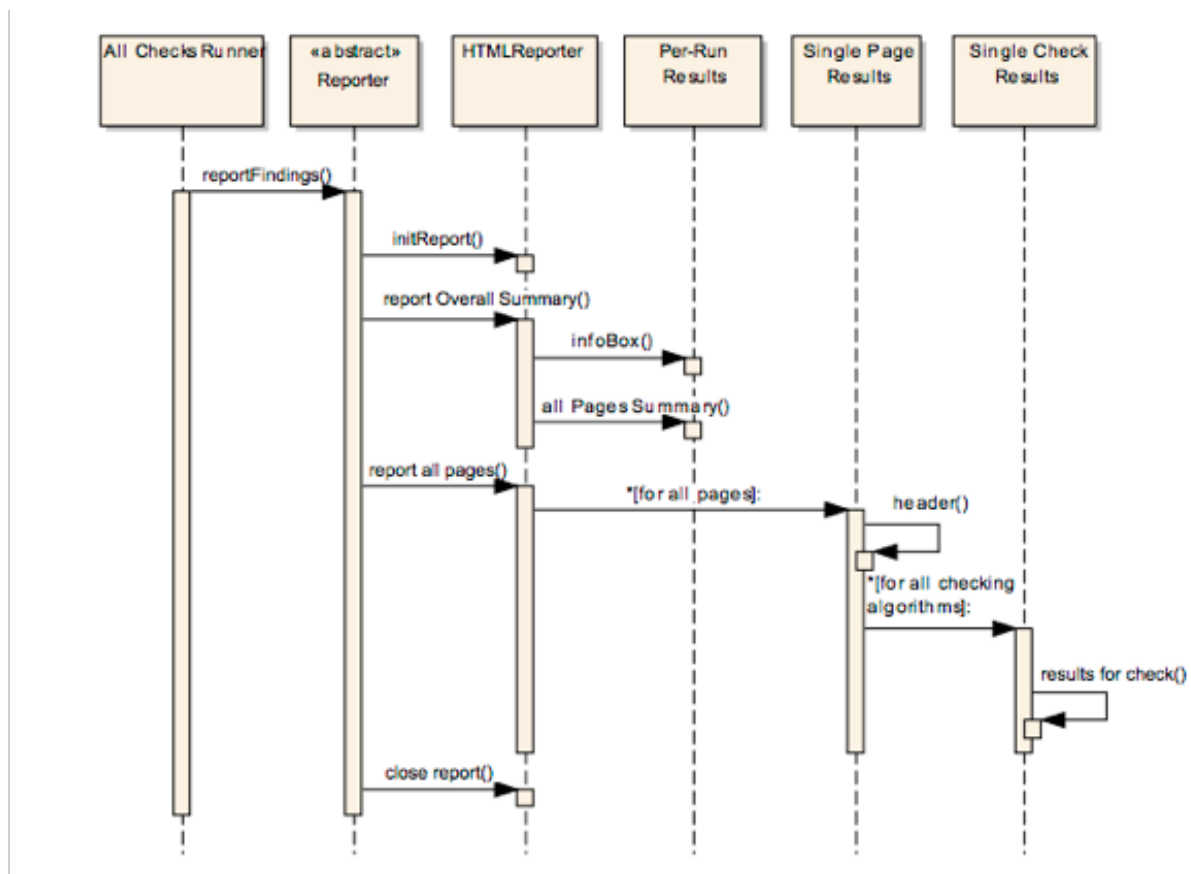
A typical scenario within HtmlSC is the execution of *all* available checking algorithms on a set of HTML pages.

Precondition: HtmlSC has been called from a Gradle build and is properly configured.

Scenario:

1. User or build-server calls `gradle htmlSanityCheck`
2. Gradle executes build-target `htmlSanityCheck` via the HtmlSC plugin
3. The plugin executes method `AllChecksRunner.performAllChecks` with parameter `Collection<File> filesToCheck` and `File resultsDir`
4. `performAllChecks`:
 1. creates `PerRunResults` instance
 2. creates a list of `Checker` instances, one for every available checking algorithm
5. iterate over all these `Checker` instances. Every instance:
 1. executes its own checks
 2. adds one instance of `SingleCheckResult` containing its own results.
6. `reportCheckingResultsAsHTML` creates the final report.

II.6.2 Report checking results



Sequence diagram: Report results

Reporting is done in the natural hierarchy of results (see the corresponding concept in [section 8.2.1](#) for an example report).

1. per “run” (PerRunResults): date/time of this run, files checked, some configuration info, summary of results
2. per “page” (SinglePageResults):
3. create page result header with summary of page name and results
4. for each check performed on this page create a section with SingleCheckResults
5. per “single check on this page” report the results for this particular check

Additional information

- [Section IV-6](#) contains additional tips regarding the runtime view.
- You often use runtime scenarios only to *find* or *verify* building blocks, not that much for documentation.

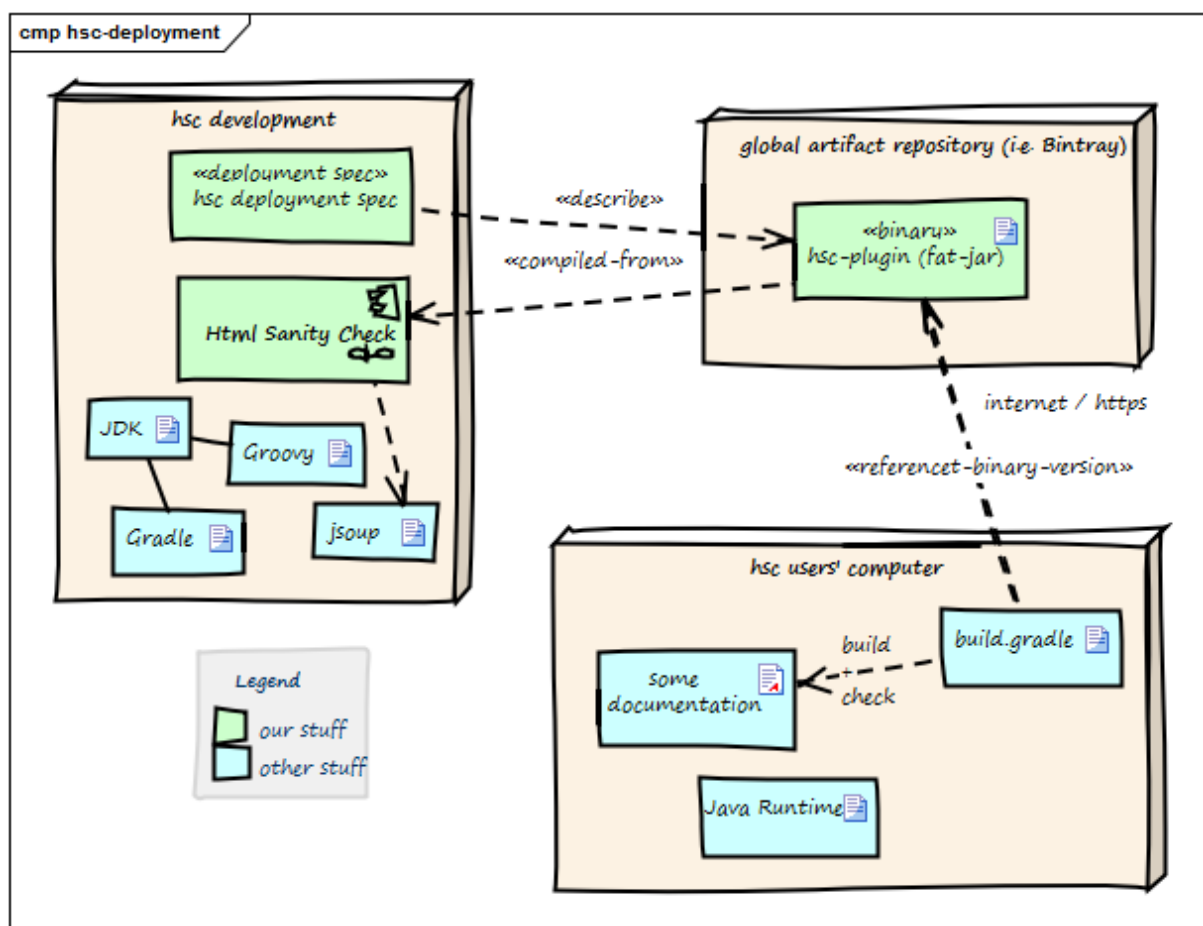
II.7 Deployment view

Content and motivation

You like to know the technical infrastructure where your system and its building blocks will be executed. That's especially important if your software is distributed or deployed on several different machines, application-servers or containers.

Sometimes you need to know about different environments (like dev, test, production).

In large commercial or web systems, aspects like scalability, clustering, automatic deployment, firewalls and load-balancing play important roles - which we definitely don't need for our small example.



HtmlSC deployment (for use with Gradle)

Node / Artifact	Description
hsc plugin binary	Compiled version of HtmlSC, including required dependencies.
hsc-development	Development environment

Node / Artifact	Description
artifact repository	Global public <i>cloud</i> repository for binary artifacts, similar to mavenCentral ²⁴ . HtmlSC binaries are uploaded to this server.
hsc user computer	Where documentation is created and compiled to HTML.
build.gradle	Gradle build script configuring (among other things) the HtmlSC plugin.

The three nodes (*computers*) shown in the diagram above are connected via Internet.

Prerequisites:

- HtmlSC developers need a Java development kit, Groovy, Gradle plus the JSoup HTML parser.
- HtmlSC users need a Java runtime (> 1.6) plus a build file named `build.gradle`. See below for a complete example.

Example for `build.gradle`

```

buildscript {
    repositories {
        maven {
            url "https://plugins.gradle.org/m2/"
        }
        jcenter()
    }
    dependencies {
        classpath (group: 'gradle.plugin.org.arc42',
                    name: 'htmlSanityCheck', version: '0.9.3')
        classpath (group: 'org.asciidoctor',
                    name: 'asciidoctor-gradle-plugin', version: '1.5.2')
    }
}

// location of AsciiDoc files
def asciidocSrcPath = "$projectDir/src/asciidoc"

// location of images used in AsciiDoc documentation
def srcImagesPath = "$asciidocSrcPath/images"

// path for asciidoc-gradle-convert
def htmlOutputPath = "$buildDir/asciidoc/html5"

// images used by generated html
def targetImagesPath = htmlOutputPath + "/images"

```

²⁴[http://search.maven.org](https://search.maven.org)

```

// where HTMLSanityCheck checking results ares stored
def checkingResultsPath = "$buildDir/report/htmlchecks"

apply plugin: 'org.asciidoctor.convert'
asciidoctor {
    sourceDir = new File( asciidocSrcPath )
    options backends: ['html5'],
            doctype: 'book', icons: 'font',
            sectlink: true, sectanchors: true
    resources {
        from( srcImagesPath )
        into targetImagesPath
    }
}

apply plugin: 'org.aim42.htmlSanityCheck'
htmlSanityCheck {
    dependsOn asciidoctor // ensure asciidoctor->html runs first

    sourceDir = new File( htmlOutputPath )

    // files to check, in Set-notation
    sourceDocuments = [ "many-errors.html", "no-errors.html" ]

    // where to put results of sanityChecks...
    checkingResultsDir = new File( checkingResultsPath )
}

```

Additional information

- [Section IV-7](#) contains additional tips regarding the deployment view.
- Sometimes it's useful to describe high-level deployment or infrastructure in the technical context in arc42 section 3.2, corresponding tips are given in this book

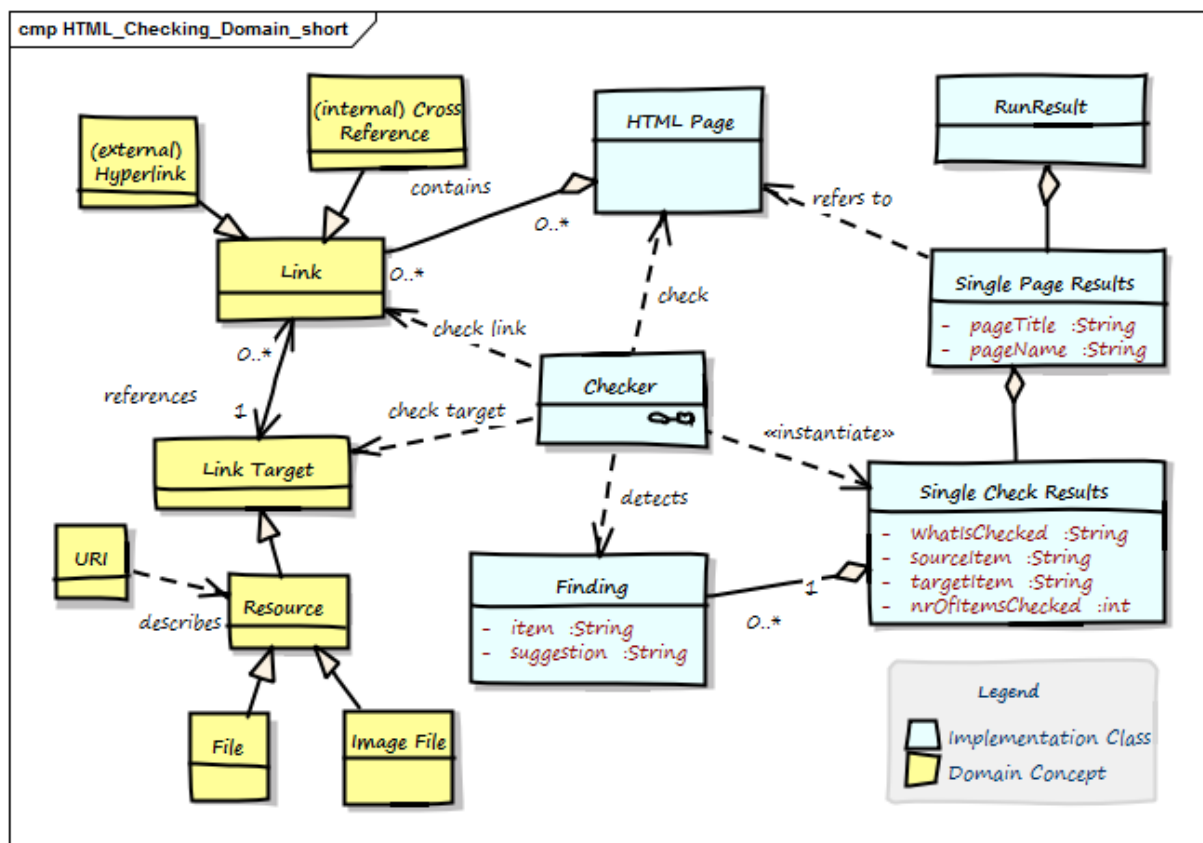
II.8 Technical and Crosscutting Concepts

Content and Motivation

You should explain crosscutting and ubiquitous rules of your system. arc42 calls them *concepts*: They often affect multiple building blocks and are relevant in several parts of the system and its implementation. Examples include:

- Rules for the usage of technologies and/or frameworks
- Implementation rules, design or architecture patterns used

8.1 Domain Model



HTML Checking Domain Model

Term	Description
Anchor	Html element to create ->Links. Contains link-target in the form <code></code>
Cross Reference	Link from one part of the document to another part within the same document. Special form of ->Internal Link, with a ->Link Target in the same document.
External Link	Link to another page or resource at another domain.
Finding	Description of a problem found by one ->Checker within the ->Html Page.
Html Element	HTML pages (documents) are made up by HTML elements .e.g., <code></code> , <code></code> and others. See the definition from the W3-Consortium ²⁵
Html Page	A single chunk of HTML, mostly regarded as a single file. Shall comply to standard HTML syntax. Minimal requirement: Our HTML parser can successfully parse this page. Contains ->Html Elements. Synonym: <i>Html Document</i> .
id	Identifier for a specific part of a document, e.g. <code><h2 id="#someHeader"></code> . Often used to describe ->Link Targets.
Internal Link	Link to another section of the same page or to another page of the same domain. Also called ->Cross Reference or <i>Local Link</i> .
Link	Any a reference in the ->Html Page that lets you display or activate another part of this document (->Internal Link) or another document, image or resource (can be either ->Internal (local) or ->External Link). Every link leads from the <i>Link Source</i> to the <i>Link Target</i> .
Link Target	Target of any ->Link, e.g. heading or any other a part of ->Html Documents, any internal or external resource (identified by URI). Expressed by ->id.
Local Resource	local file, either other Html files or other types (e.g. pdf, docx)
Run Result	The overall results of checking a number of pages (at least one page).
Single Page Result	A collection of all checks of a single ->Html Page.
URI	Universal Resource Identifier. Defined in RFC-2396 ²⁶ , the ultimate source of truth concerning link syntax and semantic.

8.2 Structure of HTML Links

Remark: For many web developers or HTML experts the following information on URI syntax might be completely evident. As we wrote this book also for different kind of people, we included this information anyhow.

HtmlSC performs various checks on HTML links (hyperlinks), which usually follow the URI syntax specified by [RFC-2396](#)²⁷. URIs are generally used to link to arbitrary resources (documents, files or parts within documents).

Their general structure is depicted in the following figure - you also find a unit test below.

²⁵http://www.w3schools.com/html/html_elements.asp

²⁶<http://www.ietf.org/rfc/rfc2396.txt>

²⁷<http://www.ietf.org/rfc/rfc2396.txt>

`[protocol://][host][:port][path][?query][#ref]`
`http://example.com:42/docs/index.html?name=aim42#INTRO`

Figure: Generic URI structure

Test showing generic URI syntax

```

@Test
public void testGenericURISyntax() {
    // based upon an example from the Oracle(tm) Java tutorial:
    // http://docs.oracle.com/javase/tutorial/networking/urls/urlInfo.html
    def aURL = new URL(
        "http://example.com:42/docs/tutorial/index.html?name=aim42#INTRO");
    aURL.with {
        assert getProtocol() == "http"
        assert getAuthority() == "example.com:42"
        assert getHost() == "example.com"
        assert getPort() == 42
        assert getPath() == "/docs/tutorial/index.html"
        assert getQuery() == "name=aim42"
        assert getRef() == "INTRO"
    }
}

```

8.3 Multiple Checking algorithms

HtmlSC uses the [template-method-pattern](http://sourcemaking.com/design_patterns/template_method/)²⁸ to enable flexible checking algorithms:

“The Template Method defines a *skeleton of an algorithm* in an operation, and defers some steps to subclasses”.

We achieve that by defining the skeleton of the checking algorithm in one operation (per `formCheck`), deferring the specific checking algorithm steps to subclasses. The invariant steps are implemented in the abstract base class, while the variant checking algorithms have to be provided by the subclasses.

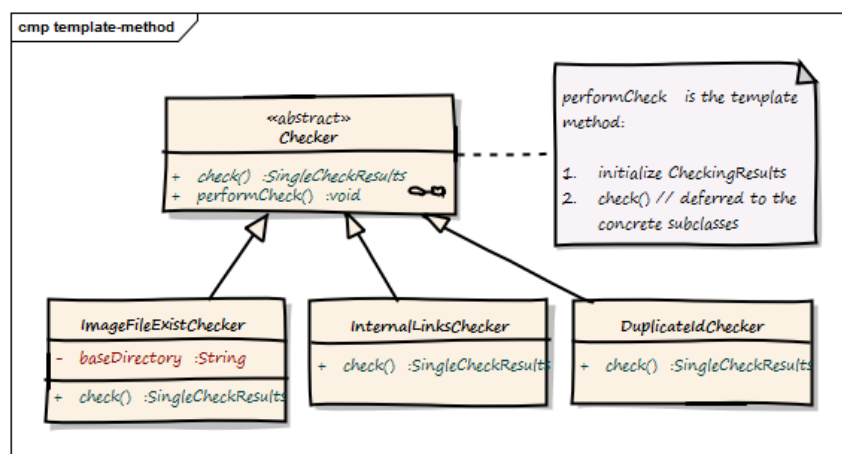
²⁸http://sourcemaking.com/design_patterns/template_method/

Template method for performing a single type of checks

```

/**
 * Prerequisite: pageToCheck has been successfully parsed,
 * prior to constructing this Checker instance.
 */
public CheckingResultsCollector performCheck() {
    // assert prerequisite
    assert pageToCheck != null
    initResults()
    return check() // subclass executes the actual checking algorithm
}

```



Template Method (excerpt)

Component	Description
Checker	<i>abstract</i> base class, containing the template method <code>check()</code> plus the public method <code>performCheck()</code>
Image File ExistC Checker	checks if referenced local image files exist
Internal Links Checker	checks if cross references (links referenced within the page) exist
DuplicateIdChecker	checks if any id has multiple definitions

8.4 Reporting

HtmlSC supports the following output (== reporting) formats and destinations:

- formats (HTML and text) and
- destinations (file and console)

The reporting subsystem uses the template method pattern to allow different output formats (e.g. Console and HTML). The overall structure of reports is always the same.

The (generic and abstract) reporting is implemented in the abstract Reporter class as follows:

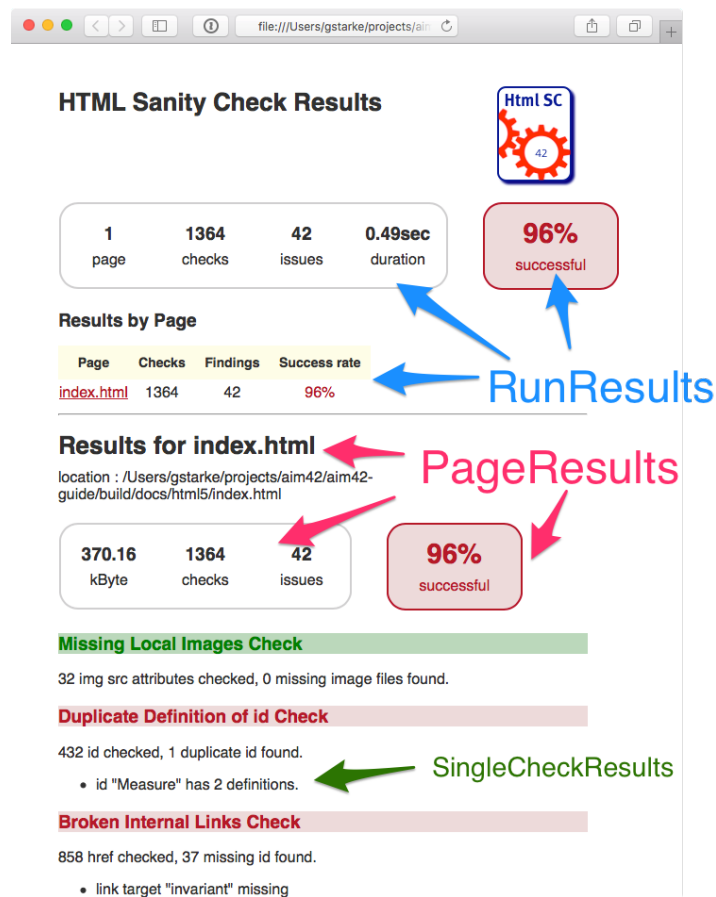
Report findings using TemplateMethod pattern

```
/**
 * main entry point for reporting - to be called when a report is requested
 * Uses template-method to delegate concrete implementations to subclasses
 */
public void reportFindings() {
    initReport()           // (1)
    reportOverallSummary() // (2)
    reportAllPages()       // (3)
    closeReport()          // (4)
}

private void reportAllPages() {
    pageResults.each { pageResult ->
        reportPageSummary( pageResult )           // (5)
        pageResult.singleCheckResults.each { resultForOneCheck ->
            reportSingleCheckSummary( resultForOneCheck ) // (6)
            reportSingleCheckDetails( resultForOneCheck ) // (7)
            reportPageFooter()
        }
    }
}
```

1. initialize the report, e.g. create and open the file, copy css-, javascript and image files.
2. create the overall summary, with the overall success percentage and a list of all checked pages with their success rate.
3. iterate over all pages
4. write report footer - in HTML report also create back-to-top-link
5. for a single page, report the number of checks and problems plus the success rate
6. for every singleCheck on that page, report a summary and
7. all detailed findings for a singleCheck.
8. for every checked page, create a footer, page break or similar to graphically distinguish pages between each other.

The sample report below illustrates this.



Sample report showing run/page/check hierarchy of results

Additional information

- [Section IV-8](#) contains additional tips regarding crosscutting concepts.

II.9 Design Decisions

Content and Motivation

You like to understand important, huge, expensive, risky or otherwise special architecture and design decisions.

It's especially interesting to keep the *reasons* for these decisions.

9.1 Checking of external links postponed

In the current version of HtmlSC we won't check external links. These checks have been postponed to later versions.

9.2 HTML Parsing with jsoup

To check HTML we parse it into an internal (DOM-like) representation. For this task we use [Jsoup](#)²⁹, an open-source parser without external dependencies.

To quote from their website:

[quote] jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jQuery-like methods.

Goals of this decision: Check HTML programmatically by using an existing API that provides access and finder methods to the DOM-tree of the file(s) to be checked.

Decision Criteria:

- few dependencies, so the HtmlSC binary stays as small as possible.
- accessor and finder methods to find images, links and link-targets within the DOM tree.

Alternatives:

- HTTPUnit: a testing framework for web applications and -sites. Its main focus is web testing and it suffers from a large number of dependencies.
- jsoup: a plain HTML parser without any dependencies (!) and a rich API to access all HTML elements in DOM-like syntax.

Find details about usage of this parser in the [HTML encapsulation](#).

²⁹<http://jsoup.org>

9.3 String Similarity Checking using Jaro-Winkler-Distance

The small [java string similarity library](#)³⁰ (by Ralph Allen Rice) contains implementations of several similarity-calculation algorithms. As it is not available as public binary, we use the sources instead, primarily: `net.ricecode.similarity.JaroWinklerStrategy`.

Additional information

- [Section IV-9](#) contains additional tips regarding decisions.

³⁰<https://github.com/rrice/java-string-similarity>

II.10 Quality Scenarios

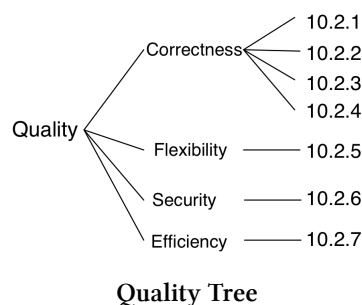
Content and motivation

You want to know specific and operational quality requirements, at best in form of a specific *quality tree*. This section completes, augments or refines the top-quality goals already given in arc42-section 1.2.

The quality tree from section 10.1 shows the quality attributes which are significant for your system, refined by scenarios.

Remark: For our small example, such a quality tree is overly extensive... whereas in real-live systems we've seen quality trees with more than 100 scenarios.

10.1 Quality tree



10.2 Quality Scenarios

ID	Description
10.2.1	Every broken internal link will be found.
10.2.2	Every missing (local) image will be found.
10.2.3	Correctness of all checks is ensured by automated positive and negative tests.
10.2.4	The results-report must contain <i>all</i> results (aka findings)
10.2.5	HtmlSC shall be extensible with new checking algorithms and new usage scenarios (i.e. from different build systems)
10.2.6	HtmlSC leaves its source files completely intact: Content of files to be checked will <i>never</i> be modified.
10.2.7	HtmlSC performs all checks on a 100kByte HTML file in less than 10 seconds.

Additional information

- The most important quality goals (expressed as *scenarios*) have been summarized in the introductory [section IV-1.2](#).

- [Section IV-10](#) contains additional tips regarding quality scenarios.

II.11 Risks and technical debt

Content and Motivation

You want to know the technical risks of your system, so you can address potential future problems.

In addition you want to support your management stakeholders (i.e. project management, product owner) by identifying technical risks.

Remark: In our small example we don't see any *real* risks for architecture and implementation. Therefore the risks shown below are a bit artificial...

11.1 Technical risks

Risk	Description
Bottleneck with access rights on public repositories	Currently only one single developer has access rights to deploy new versions of HtmlSC on public servers like Bintray or Gradle plugin portal.
High effort required for new versions of AsciiDoc	Upgrading AsciiDoc from v-0.x to v-1.x required significant effort for HtmlSC due to several breaking changes in AsciiDoc. Such high effort might be needed again for future upgrades of the AsciiDoc API

11.2 Business or domain risks

Risk	Description
System might become obsolete	In case AsciiDoc or Markdown processors implement HTML checking natively, HtmlSC might become obsolete.

Additional information

- [Section IV-11](#) contains additional tips regarding risks.
- Help your management by informing them about technical risks - and options for their mitigation.

II.12 Glossary

Content and Motivation

You need to understand the most important domain terms and expressions, that stakeholders use when communicating about the system, its interfaces, goals and requirements.

The glossary is one manifestation of our pledge for “*explicit, not implicit*”, as it associates words or terms with sense, meaning and semantics.

In the case of our small example, the terms given here should be good friends to most developers. You find a more interesting version of the glossary in [section II-8.1](#).

Term	Definition
Link	A reference within an \rightarrow HTMLPage. Points to \rightarrow LinkTarget
Cross Reference	Link from one part of a document to another part within the same document.
External Hyperlink	Link to another HTML-page or to a resource within another domain or site.
Run Result	Combined checking results for multiple pages (\rightarrow HTMLPages)
SinglePageResults	Combined results of all Checker instances for a single HTML page.

Additional information

- [Section IV-12](#) contains additional tips regarding the glossary.
- Many terms in the glossary correspond to parts of the “Ubiquitous Language” from Eric Evans’ [Domain Driven Design](#)³¹

³¹<https://domainlanguage.com/dd/>

It's not the end, is it?

This is the end of the sample book - but not the end of arc42 tips. If you liked what you read, please support arc42 and the authors of this book (Gernot Starke and Peter Hruschka) by:

- getting the complete book from [Leanpub](https://leanpub.com/arc42inpractice)³²,
- telling your friends and colleagues about it or
- [tweeting about](https://twitter.com/home?status=https://leanpub.com/arc42inpractice%20@gernotstarke)³³

Thank you!

³²<https://leanpub.com/arc42inpractice>

³³<http://twitter.com/home?status=https://leanpub.com/arc42inpractice%20@gernotstarke>