# arc42 Overview

## 1. Introduction and Goals

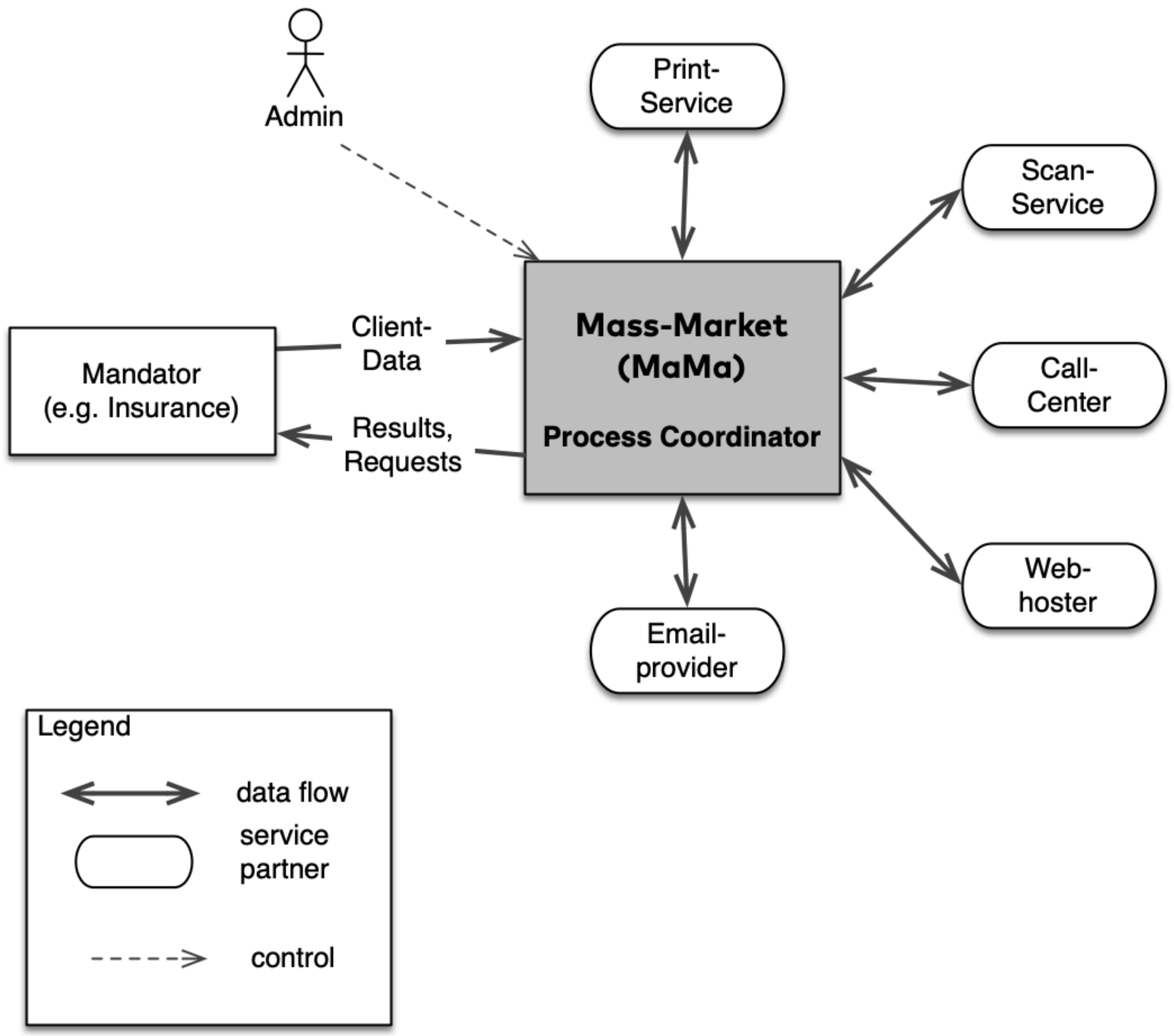
- underlying business goals, essential features and functional requirements for the system
- quality goals for the architecture
- relevant stakeholders and their expectations

## 2. Constraints

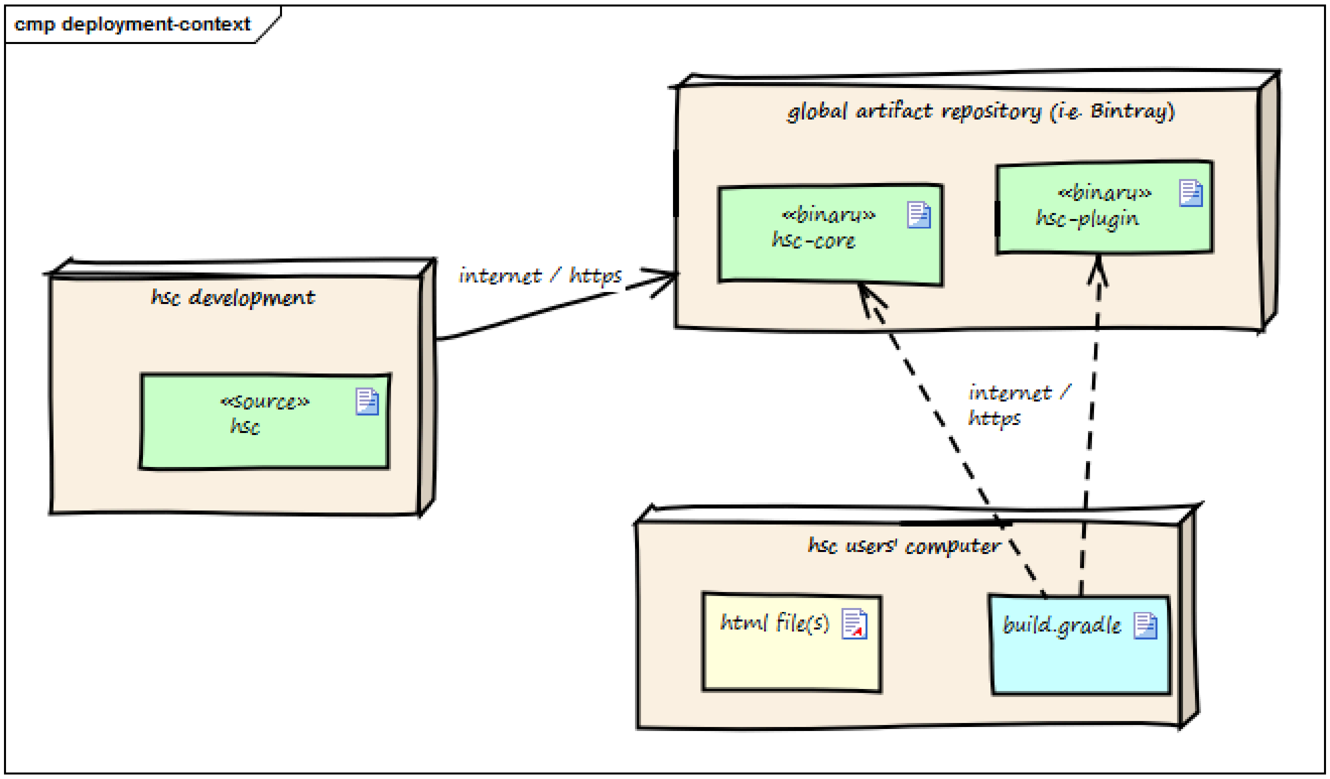
- Constraints which are limiting in terms of e.g., programming.
- **Example** for a HTML Sanity Checker:
  - platform-independent and should run on the major operating systems (Windows™, Linux, and Mac-OS™)
  - integrated with the Gradle build tool
  - runnable from the command line
  - developed under a liberal open-source license

## 3. Context and Scope

- Specify external interfaces
- in our case:
  - Stripe
  - E-Mail Service
  - SQL Database
- **Examples**:
  - Business Context



- Technical Context

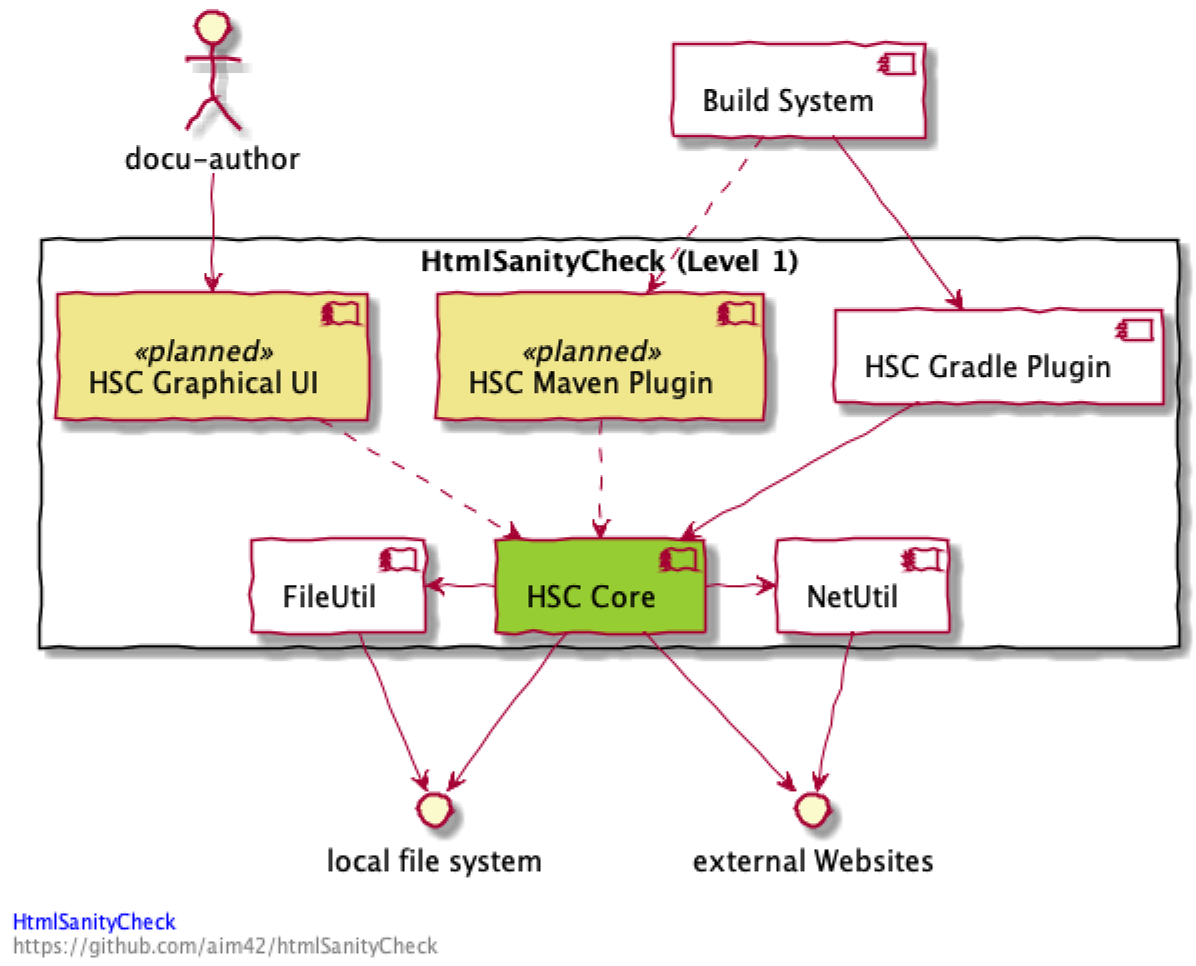


## 4. Solution Strategy

- technology decisions
- decisions about the top-level decomposition of the system, e.g. usage of an architectural pattern or design pattern
- decisions on how to achieve key quality goals
- relevant organizational decisions, e.g. selecting a development process or delegating certain tasks to third parties.

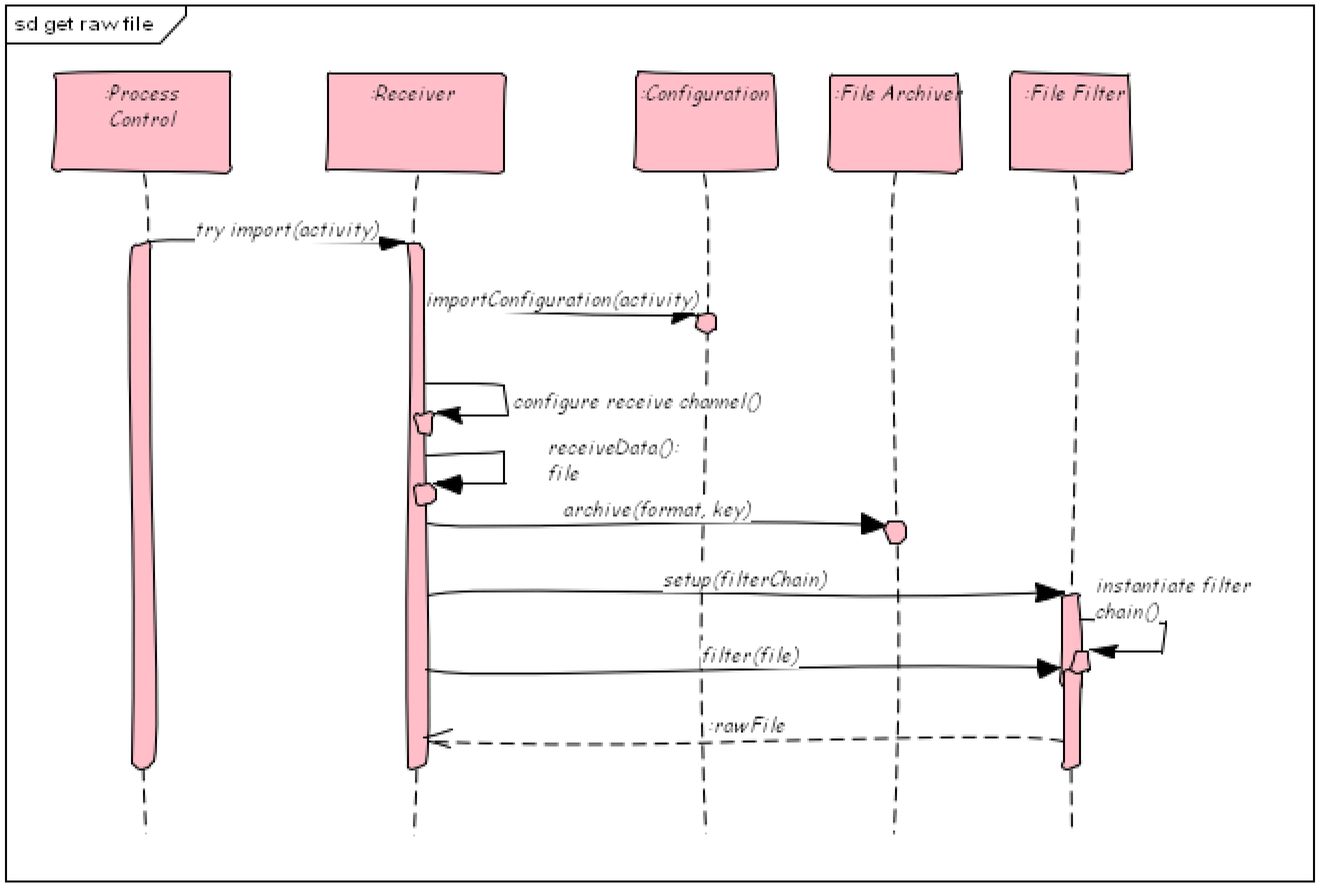
## 5. Building Block View

- Decomposition of system into building blocks (modules, components, classes)
- **Example**:



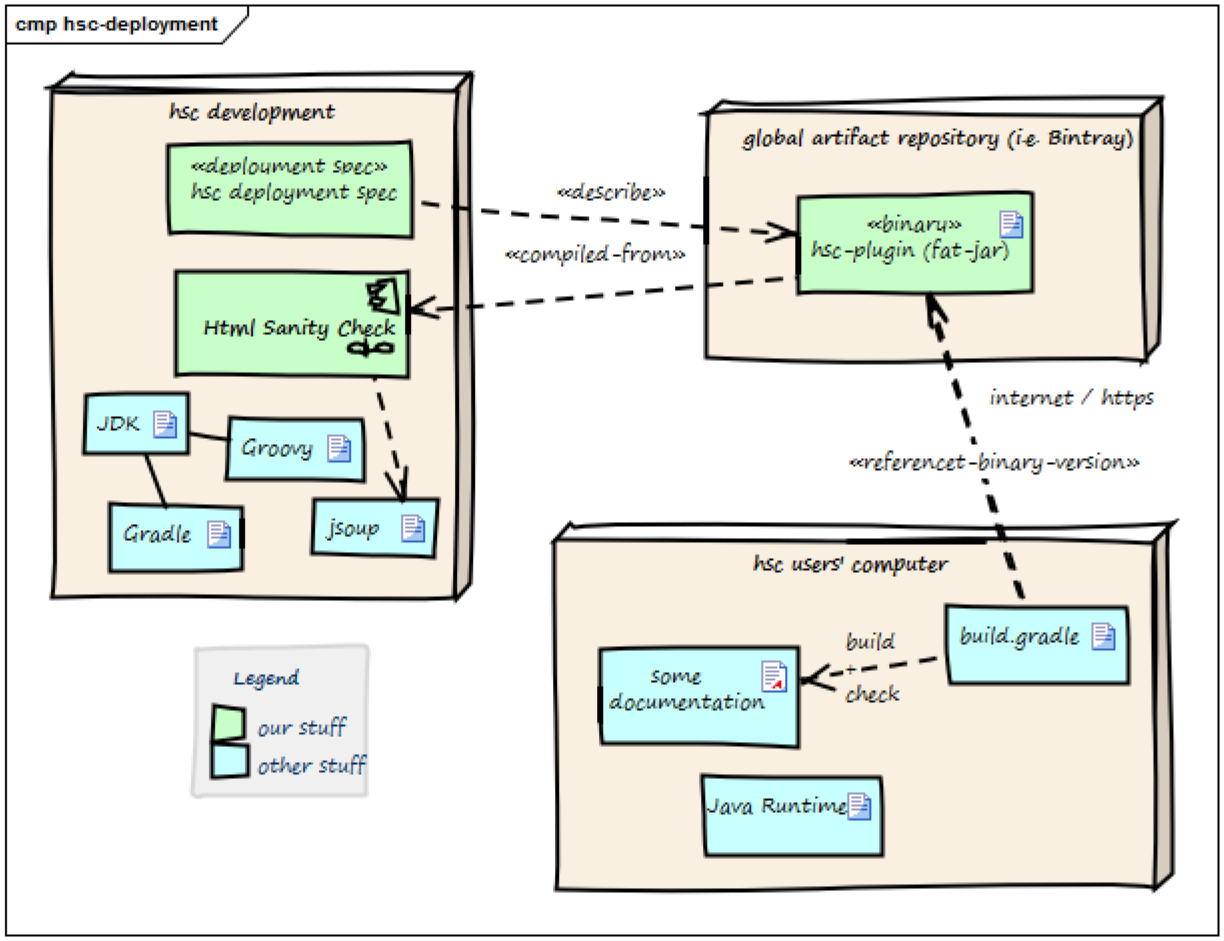
# 6. Runtime View

- important use cases or features: how do building blocks execute them?
- interactions at critical external interfaces: how do building blocks cooperate with users and neighbouring systems?
- operation and administration: launch, start-up, stop
- error and exception scenarios
- **Example**:



## 7. Deployment View

- the technical infrastructure used to execute your system, with infrastructure elements like geographical locations, environments, computers, processors, channels and net topologies as well as other infrastructure elements and
  the mapping of (software) building blocks to that infrastructure elements.
- **AZURE!**
- Probably put Azure Architecture Diagrams here
- **Example**



# 8. Crosscutting Concepts

This section describes overall, principal regulations and solution ideas that are relevant in multiple parts (→ cross-cutting) of your system. This may include:

- domain models
- architectural patterns or design patterns
- rules for using specific technology
- principale, often technical decisions of overall decisions
- implementation rules
- **Example**:
    - Within a system, a common format for log-messages shall be established, combined with a common convention of choosing the appropriate log-destination. These decisions, along with implementation examples, could be described as “logging-concept”.

# 9. Architectural Decisions

- Important, expensive, large scale or risky architecture decisions including rationales. With "decisions" we mean selecting one alternative based on given criteria.
- **Example**:
  - 9.2 HTML Parsing with jsoup
    - To check HTML we parse it into an internal (DOM-like) representation. For this task we use Jsoup, an open-source parser without external dependencies.
    - 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 easily locate 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.

# 10. Quality Requirements

- Quality requirements of lesser priority (the important ones were already described in Chapter 1: Introduction and Goals!)
- **Example**:
  - 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.

# 11. Risks and Technical Debt

- A list of identified technical risks or technical debts, ordered by priority
- **Examples**:
  - 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.
  - System relies on gradle - which may not be available on target computers

        - Although the Java Runtime is installed on many computers, it might not be available for every potential HtmlSC user.
    - System might become obsolete
        - In case AsciiDoc or Markdown processors implement HTML checking natively, HtmlSC might become obsolete.

## 12. Glossary

- Documenting important terms
    - E.g., What is a CSP?
    - **Example**:
        - Cross Reference:
            - Link from one part of a document to another part within the same document.
        - Run Result:
            - Combined checking results for multiple pages (HTMLPages)

# Assignment

- 5-6 pages per person
- ***(Vinay)***:
    - 1- Introduction and Goals
    - 2- Constraints
- ***(Dennis, Domi, Jatender)***:
    - 3- Context and Scope *(Domi)*
    - 4- Solution Strategy *(Dennis, Jatender)*
    - 5- Building Block View (Frontend side) *(Dennis, Jatender)*
- ***(Domi, Edward)***:
    - 5- Building Block View (Backend side)
    - 6- Runtime View
    - 7- Deployment View
- ***(Dennis, Jatender)***:
    - 8- Crosscutting Concepts
    - 9- Architectural Decisions
- ***(Domi, Edward)***:
    - 10- Quality Requirements
    - 11- Risks and Technical Debt

- **Everybody is responsibe to keep 12- Glossary up to date with their additions!**

## Further Notes

- Deadline for us: Friday, 28th at 6 p.m.
- At the bottom of document, attach link to our GitHub, say which GitHub user is who, and write "We consider commit with commit id 234141414141431... the final commit for the code base. Any commits after that should only change the documentation (this document) and presentation slides"