openIMIS

Gumzo ya Mwezi 07/10/2019

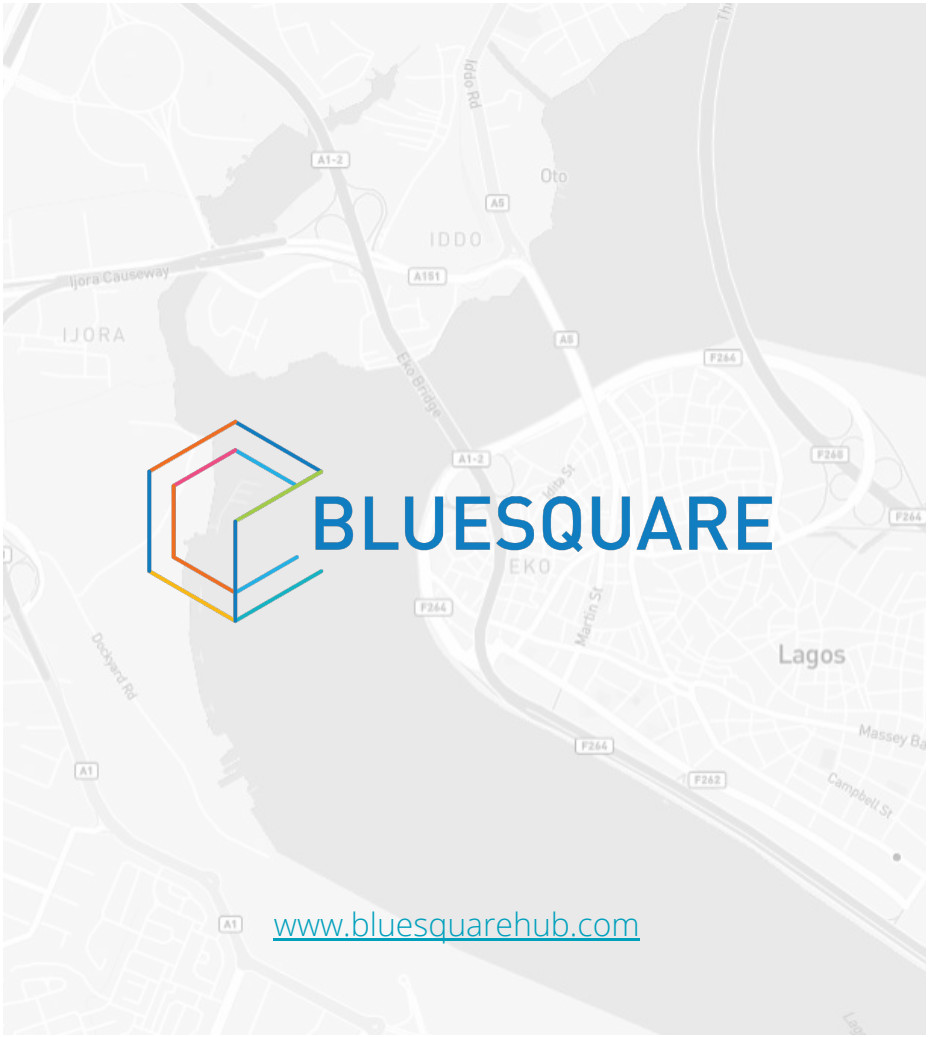
Agenda

- Bluesquare:
  - who we are
  - our engagement towards openIMIS community
  - our methodology
- Achieved

  what we delivered this month

- Roadmap

  what we will deliver and what are our dependencies



Delivering innovative technology for better lives.

www.bluesquarehub.com

Fall 2018

what we do

COUNTRY LEVEL DATA SYSTEMS 24 COUNTRIES

We build technologies that enhance governmental health data systems with a focus on three markets:

HEALTH FINANCING DATA SYSTEMS

- Data systems for purchasers, health insurance, Ministries of Health
- Example: Develop a Pay for Performance data system in Kyrgyz hospitals

GOVERNMENT HEALTH DATA WAREHOUSES

- Example: The health data warehouse in Morocco

DISEASE OR THEMATIC DATA SYSTEMS

- Diabetes
- HIV
- Tuberculosis
- Malaria
- Immunization systems
- Vector Borne eradication systems (i.e. sleeping sickness)
- Family Planning
- Emergency Obstetric Care

Bluesquare develops these data systems based on a suite of in-house software products connected to DHIS2 a popular open source data management platform used by over 40 governments.

How we do
IT products and data services

We deliver technologies and services that strengthen governmental health data systems, mainly:

Hesabu (aka ORBF)

- An open sourced rule engine that allows complex calculations in DHIS2, a popular open source data management platform. This is particularly useful for health financing data systems.

Data Viz

- A public web dashboard that allows showcasing results.

Modeling and data science

- Statistical analysis, Data cleaning, Modeling & machine learning and analysis automation to help customers bring value out of their health data.

Bluesquare suite of in-house software products and services allow collecting, computing, analyzing and visualizing data in a intelligent and friendly manner.


24
COUNTRIES

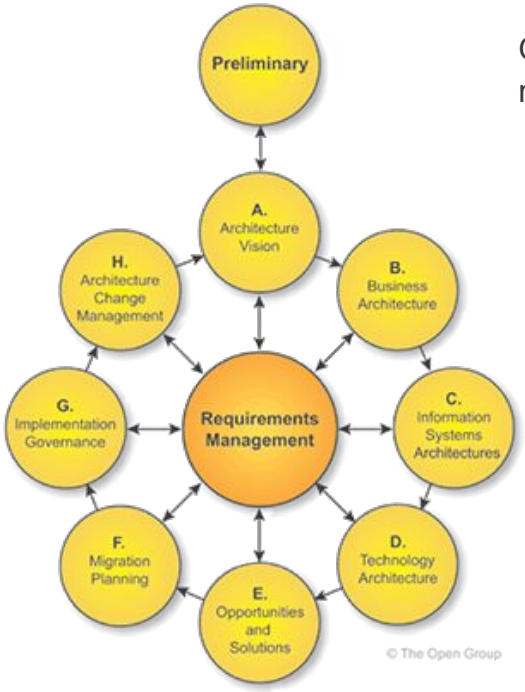
Bluesquare: our engagement towards openIMIS community

We believe that health insurance will be at the heart of the UHC agenda in many countries.

openIMIS modular transformation is an opportunity to develop code that can be used at scale to help provide health services “for the global good” (i.e. exact DNA of Bluesquare).

Creating synergies with our existing and future health-financing portfolio, promoting the tool in the countries where we operate.

Bluesquare: our methodology



Our approach to deliver the openIMIS modules borrows several concepts from TOGAF, most important one being the ADM (Architecture Development Method):

- Iterative, ensuring pragmatism and responsiveness in delivered solution
- We strive to keep things simple: we aim to use the TOGAF framework as a guide not a rule book. Where we feel it will serve this project we will make use of it. However, our proposed approach is much lighter than a traditional TOGAF implementation effort.
- It helps any community member to find/contribute to the appropriate part of the system.

Agenda

- Bluesquare:
  - who we are
  - our engagement towards openIMIS community
  - our methodology
- Achieved

  what we delivered this month

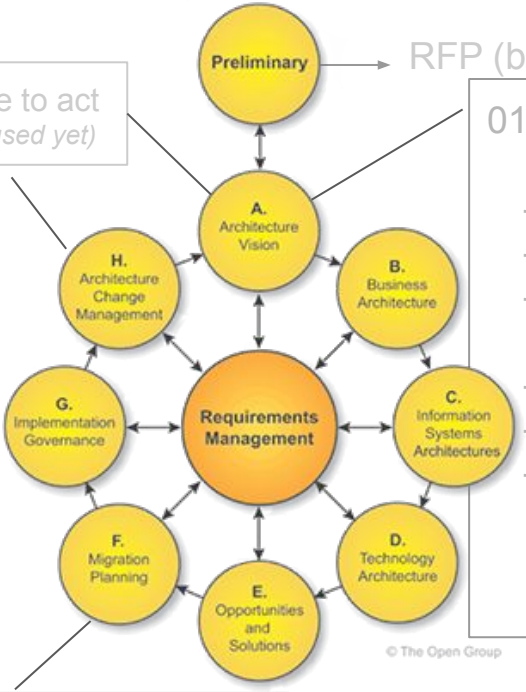
- Roadmap

  what we will deliver and what are our dependencies

Iteration 1: (55 + 1) / 60

Achieved (Iteration 1): FHIR API

No architecture change to act
(but deliverable not really used yet)



RFP (by GIZ) - 10/2018

01/2019: start of work

- Brainstorming (blsq internal)
- Conceptual architecture documentation (in openIMIS wiki)
- Architecture Presentations (16/01 and 23/01 + follow-up calls)

6 m/d

- Technology stack proposal
- Migration strategy & roadmap
- PoC on proposed technology stack
  (preparation 'hands on' session at Bonn Workshop)

14 m/d

Under progress:

Objective iteration 1: FHIR API

03/2019 13 m/d
04/2019: 12 m/d
05/2019: 4 m/d
06/2019: 0,5 m/d
07/2019: 0 m/d
08/2019: 1,5 m/d
09/2019: 1 m/d

… follow-up ONCO-98 (Eligibility Validity Period is not included in Response of Eligibility Response), and release 0.0.5 FHIR API

TODO (Bluesquare):
follow up, bug fix and adaptations

Bonn Workshop
- Demonstrated technology stack and migration strategy
- Agenda alignments

4 m/d

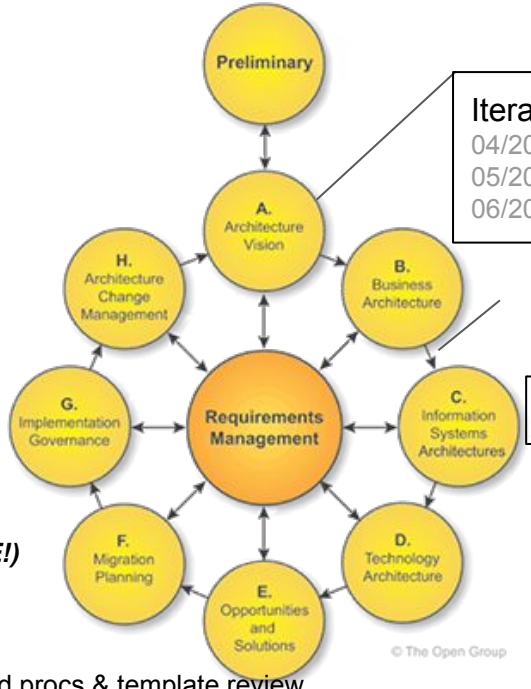
Achieved (Iteration 2): All Proxies

Under maintenance:
Objective iteration 1: All proxy (frontend) setup
05/2019: 10 m/d
06/2019: 8,5 m/d
07/2019: CLOSED
Enquiry screen moved into the claim module scope!

01/05 - No change in Iteration 2 scope (architecture vision, business arch, …)

CLOSED
Since 01/08

Achieved (Iteration 3): Claim Module



Iteration 3:

04/2019 : Claim module scope document (draft)	2 m/d
05/2019 : Mapping attempt to JLN business processes	0.5 m/d
06/2019 : Claim main screen mockups	4,5 m/d

06/2019: Django Dynamic Rest > (GraphQL) 4 m/d

07/2019: 12 m/d (scope, design & Enquiry)
- Enquiry Screen
- Claim scope & Design

08/2019: 34 m/d (Claim implementation)

09/2019: 29 m/d (Claim refinement)
- Available (6/09) on bluesquare test server
  … review and adaptations
- Nepali calendar in frontend, translation platform,
  user auto-provisioning,...
- Submit & Process stored procs ported to py ***(!HUGE!)***
- Reports (with a 'bulk' default template)
- Testing,… (OMT tickets)

TODO Bluesquare (remains 4 m/d)
- Batch *(relative prices)* & Reports *(pbh, pbc & pbp)* backend stored procs & template review
- Some missing Nepali customizations (claims highlights on amount/same insuree, claim attachments)
- Security
- Testing, testing, testing… (especially the claim validations!)

>> We will clearly be over budget (+/-20m/d) : will be consuming contingency… and even transfer 10 m/d budget from other iterations !!

Claim Validations (@Submit and @Process)

- Uses Gregorian calendars
- Single function > 1000 lines of code
  - Split into specific validations
  - Ability for implementers to compose an alternate validation based on provided methods
- Lots of repetition
  - Python allows more flexible/readable code
- Testing
  - Each validation can be tested and avoid regression
- More specific errors
  - Claim item xxxx, provided xxxx over maximum number allowed xxxx
- Discrepancies between Submit and Process

Claim Validations (@Submit and @Process)

```
# Then check IP deductibles
if not deductible:
    if (product.ceiling_interpretation_id == 'I' and hospitalization == 1) or\
        (product.ceiling_interpretation_id == 'H' and hf_level == 'H'):
        # Hospital IP
        ded_ip = _get_dedrem("ded_ip", "I", "ded_ip", product, claim, claimitem)
        deductible = ded_ip if ded_ip else deductible
        # if product.ded_ip_treatment:
        #     deductible = product.ded_ip_treatment
        #     deductible_type = 'I'
        #     prev_deducted = 0
        # if product.ded_ip_insuree:
        #     deductible = product.ded_ip_insuree
        #     deductible_type = 'I'
        #     prev_remunerated = ClaimDedRem.objects\
        #         .filter(policy_id=claimitem.policy_id)\
        #         .filter(insuree_id=claim.insuree_id)\
        #         .exclude(claim_id=claim.id)\
        #         .aggregate(Sum("ded_ip"))
        # if product.ded_ip_policy:
        #     deductible = product.ded_ip_policy
        #     deductible_type = 'I'
        #     prev_remunerated = ClaimDedRem.objects\
        #         .filter(policy_id=claimitem.policy_id)\
        #         .exclude(claim_id=claim.id)\
        #         .aggregate(Sum("ded_ip"))
    else:
        # Hospital OP
        ded_op = _get_dedrem("ded_op", "O", "ded_op", product, claim, claimitem)
        deductible = ded_op if ded_op else deductible
        # if product.ded_op_treatment:
        #     deductible = product.ded_op_treatment
        #     deductible_type = 'O'
        #     prev_deducted = 0
```

```
IF @BaseCategory <> 'V'
    BEGIN
        IF (ISNULL(@CeilingSurgery, 0) > 0) AND @BaseCategory = 'S' --  Ceiling check for Surgery
            BEGIN
                IF @WorkValue + @PrevRemuneratedSurgery + @RemuneratedSurgery <= @CeilingSurgery
                    BEGIN
                        --we are still under the ceiling for hospitalization and can be fully covered
                        SET @RemuneratedSurgery = @RemuneratedSurgery + @WorkValue
                    END
                ELSE
                    BEGIN
                        IF @PrevRemuneratedSurgery + @RemuneratedSurgery >= @CeilingSurgery
                            BEGIN
                                --Nothing can be covered already reached ceiling
                                SET @ExceedCeilingAmountCategory = @WorkValue
                                SET @RemuneratedSurgery = @RemuneratedSurgery + 0
                                SET @WorkValue = 0
                            END
                        ELSE
                            BEGIN
                                --claim service can partially be covered , we are over the ceiling
                                SET @ExceedCeilingAmountCategory =
                                          @WorkValue + @PrevRemuneratedSurgery + @RemuneratedSurgery -
                                          @CeilingSurgery
                                SET @WorkValue = @WorkValue - @ExceedCeilingAmountCategory
                                SET @RemuneratedSurgery = @RemuneratedSurgery + @WorkValue -- we only add
                            END
                    END
            END

        IF (ISNULL(@CeilingDelivery, 0) > 0) AND @BaseCategory = 'D' --  Ceiling check for Delivery
            BEGIN
                IF @WorkValue + @PrevRemuneratedDelivery + @RemuneratedDelivery <= @CeilingDelivery
                    BEGIN
                        --we are still under the ceiling for hospitalization and can be fully covered
                        SET @RemuneratedDelivery = @RemuneratedDelivery + @WorkValue
                    END
                ELSE
                    BEGIN
                        IF @PrevRemuneratedDelivery + @RemuneratedDelivery >= @CeilingDelivery
                            BEGIN
                                --Nothing can be covered already reached ceiling
                                SET @ExceedCeilingAmountCategory = @WorkValue
                                SET @RemuneratedDelivery = @RemuneratedDelivery + 0
                                SET @WorkValue = 0
                            END
                        ELSE
                            BEGIN
                                --claim service can partially be covered , we are over the ceiling
                                SET @ExceedCeilingAmountCategory = @WorkValue + @PrevRemuneratedDelivery
                                                                   + @RemuneratedDelivery -
                                                                   @CeilingDelivery
                                SET @WorkValue = @WorkValue - @ExceedCeilingAmountCategory
                                SET @RemuneratedDelivery = @RemuneratedDelivery + @WorkValue
                                -- we only add the value that could be covered up to the ceiling
                            END
                    END
            END
```

Claim Validations (@Submit and @Process)

```python
def validate_claim(claim) -> List[ValidationError]:
    """
    Based on the legacy validation, this method returns standard codes along with details
    :param claim: claim to be verified
    :return: (result_code, error_details)
    """
    errors = []
    errors += validate_family(claim, claim.insuree)

    if len(errors) == 0:
        errors += validate_target_date(claim)

    if len(errors) == 0:
        errors += validate_claimitems(claim)
        errors += validate_claimservices(claim)

    return errors
```

```python
def validate_claimitems(claim) -> List[ValidationError]:
    errors = []
    for claimitem in claim.items.filter(validity_to__isnull=True):
        errors += validate_claimitem_validity(claimitem)
        errors += validate_claimitem_in_price_list(claim, claimitem)
        errors += validate_claimitem_care_type(claim, claimitem)
        errors += validate_claimitem_limitation_fail(claim, claimitem)
        errors += validate_item_product_family(
            claimitem=claimitem,
            target_date=claim.target_date,
            item_id=claimitem.item_id,
            family_id=claim.insuree.family_id,
            insuree_id=claim.insuree_id,
            adult=claim.insuree.is_adult(claim.target_date)
        )

    return errors
```

Claim Validations (@Submit and @Process)

```
if total_qty_provided is None or total_qty_provided >= limit_no:
    claimitem.rejection_reason = REJECTION_REASON_QTY_OVER_LIMIT
    claimitem.save()
    errors += [ValidationError("Item %s, provided %s over maximum number allowed %s",
                               params=(item_id, total_qty_provided, limit_no),
                               code=REJECTION_REASON_QTY_OVER_LIMIT)]
```

Claim Validations (@Submit and @Process)

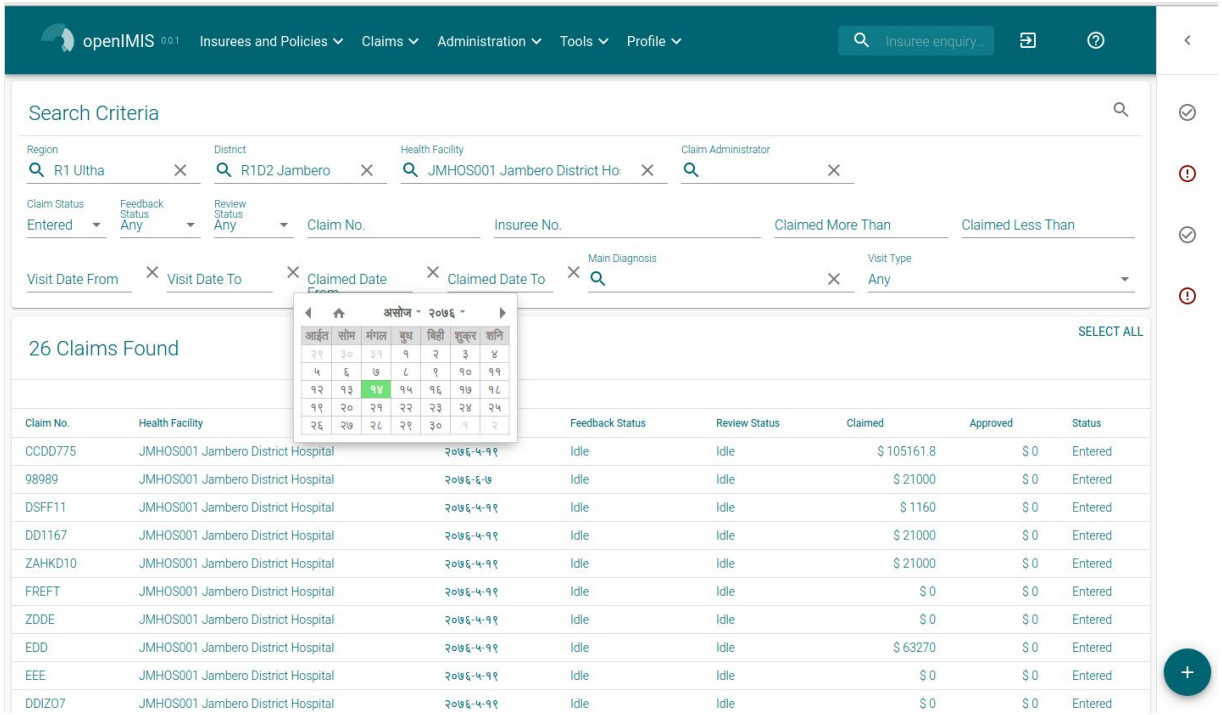
```python
def test_validate_patient_category(self):
    # When the insuree already reaches his limit of visits
    # Given
    insuree = self._create_test_insuree()
    self.assertIsNotNone(insuree)
    product = self._create_test_product("VISIT", custom_props={"max_no_visits": 1})
    policy = self._create_test_policy(product, insuree, link=True)
    service = self._create_test_service("V", custom_props={"patient_category": 1})
    product_service = self._create_test_product_service(product, service)
    pricelist_detail = self._add_service_to_hf_pricelist(service)

    # The insuree has a patient_category of 6, not matching the service category
    claim1 = self._create_test_claim({"insuree_id": insuree.id})
    service1 = self._create_test_claimservice(claim1, custom_props={"service_id": service.id})
    errors = validate_claim(claim1)

    # Then
    claim1.refresh_from_db()
    self.assertEquals(len(errors), 1)
    self.assertEquals(errors[0].code, 4)

    # tearDown
    service1.delete()
    claim1.delete()
    policy.insureepolicy_set.first().delete()
    policy.delete()
    product_service.delete()
    pricelist_detail.delete()
    service.delete()
    product.delete()
```

Claim Validations (@Submit and @Process)

- Test database is not enough to test the various scenarios
- No existing tests to validate the rewrite
- Some tests require a complex setup
- Product limitations not used in Nepal, most of @Process

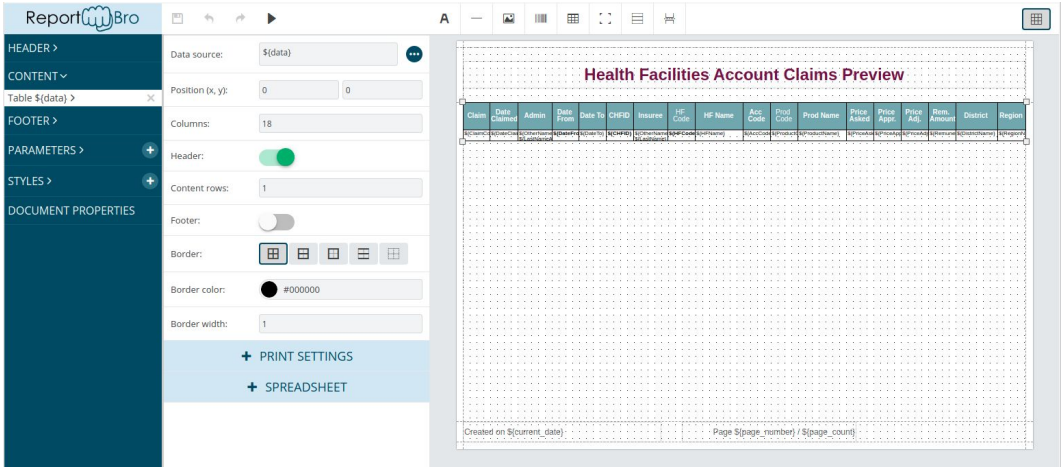
Nepali Calendar in frontend

- Front configuration vs. backend configuration
- APIs (GraphQL and FHIR) use **ISO 8601** format (and thus Gregorian dates - i.e. no 32/02/2077 !!)



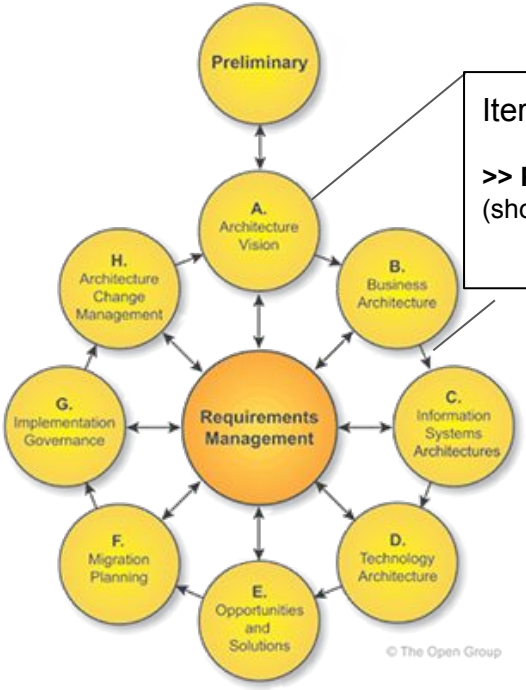
About reports

- Reports under Claim menu entry are **previews** (not the official reports, with tagging system!)
- Today report data is still provided by the StoredProcs (`uspSSRSProcessBatch` and `uspSSRSProcessBatchWithClaim`)
- The PDF rendering engine is [Report BRO](), which provides a template editor



- The used templates are loaded from the database (templates can be configured for each country via console)
- There are “defaults” layouts, but *very bulky* (will be improved)

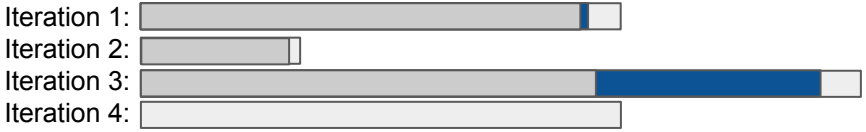
Iteration 4: Locations & Health Facilities Module



Iteration 4:

>> Proposal: [Locations & Health Facility Administration]
(should be ~ 40-50 m/d)

Roadmap (Bluesquare)



- Iteration 1 (04/2019): Building blocks for FHIR API
  - Dependencies:
    - Identified data to be mapped (cfr. xls of Soldevelo) & stored proc to be called
    - ✓Module boundaries (started with the one documented in wiki and shown @Bonn)
    - ✓follow up, bug fix and adaptations
- Iteration 2 (05/2019): "All proxy" openIMIS CLOSED
- Iteration 3 (09/2019): Claim module >> will be over budget and schedule
  - Dependencies:
    - Finalize implementation (batches, reports,...)
    - Run Acceptance Test (from input)
- Iteration 4 (11/2019): … Locations & Health Facilities