3° International Conference on
IT Data collection, Analysis and Benchmarking
Florence (Italy) - October 19, 2015

Automated Function Points in a Continuous Integration Environment (Agile AFP)

The Benefits of IFPUG Function Points in Agile Processes with Automated Open-Source QA Tools

Thimoty Barbieri, CFPS,
OCE-JPA, OCE-JSF, PMP, PMI-ACP

Contract Professor of Software Engineering at
University of Pavia
Owner at ITECH Engineering

Michele Pasquale
IT Consultant at ITECH Engineering

ITECH engineering

✓**G1.** Advocate the need to introduce Functional Sizing in Agile Metrics, to provide a stable, widely understood "normalization" factor

✓**G2.** ORM-Object Relational Mapping, an Object-Oriented Technique which encapsulates and abstracts model concepts can be leveraged to perform *transparently* automatic functional sizing

✓**G3.** We propose our early results in developing a model-transparent, open-source implementation of AFP, which could be suitable for fast ADM cycles in Continuous Integration environments

Foreword

-Why do we Need Functional Sizing in Agile?

-Why do we Need Automated Sizing in Agile?

ADM (Application Development and Management): "Software Factories" dealing with **massive amounts of code** produced by external vendors with Agile processes – need to implement a **QA approach with well-defined SLAs**, if possible in an automated way

Current common metrics (Code coverage, Tangle Index, Code Duplication, Technical Delay…) are **mostly based on LOC**. FPs are a better metric for sizing a software, because they provide a **stable**, **benchmarkable** and **comparable** **baseline**. (See CISQ AFP proposal)

However, during rapid Continuous Integration cycles, **Models** and Specifications which are used for FP counting **seldom follow the pace of development**. Moreover, it is **impractical** to perform FP counting at each Integration cycle.

Modern **ORM** (Object Relational Mapping) provides a **counting opportunity** in Agile measuring processes

Leveraging current ORM standards: using Object-Oriented Technical Specifications for counting purposes allows to include Functional Size in Agile Measurements, which is not (entirely) captured by Agile approaches like Ideal Days, Velocity, Story Points.

ORM uses Aspect Oriented Programming (AOP). This could be leveraged to **analyse code** and reverse-engineer a functional sizing measurement, **without using a manually maintained logical Model**, because the code is the model.

ORM annotations are compatible with FPA key concepts. This is a "unorthodox" use of Function Points, as they are usually counted even *before* a line of code is written, to derive expected cost/efforts on a baseline.

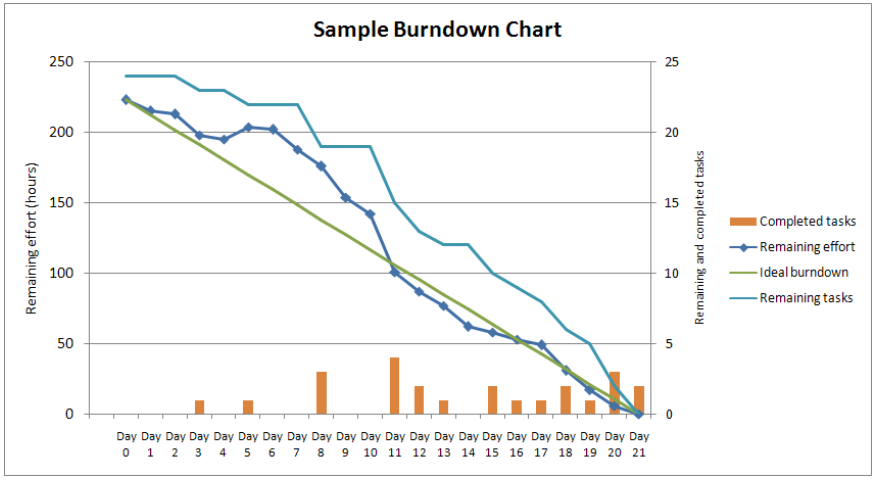
Why do we need Functional Sizing in Agile ADM?

Most used Metrics in Agile:

- **Velocity (Story Points per Sprint)** – Story Points Assessed via Delphi techniques by the Team. Criteria can change depending on Team/Project. It cannot be used to compare Sprints, worse still it cannot be used to compare/benchmark Teams or Projects!
- **Ideal Days** – Number of uninterrupted effort days, estimated by the Team per Story.

These metrics are:

- Easy to understand but Highly Subjective (by the Team)
- Geared to change with the Team, they reflect the Team's evolving "agility" and "confidence" (e.g. Velocity improves at every subsequent Sprint)

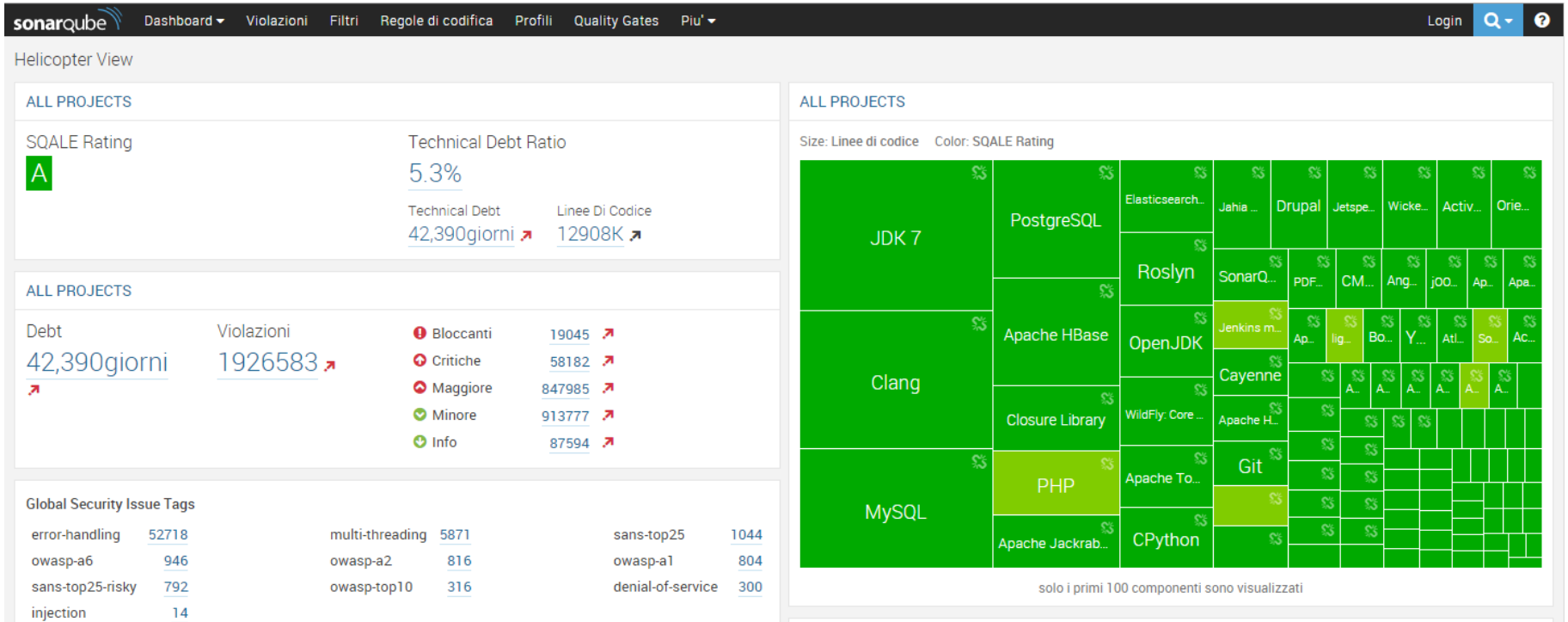


Why do we need Functional Sizing in Agile ADM?

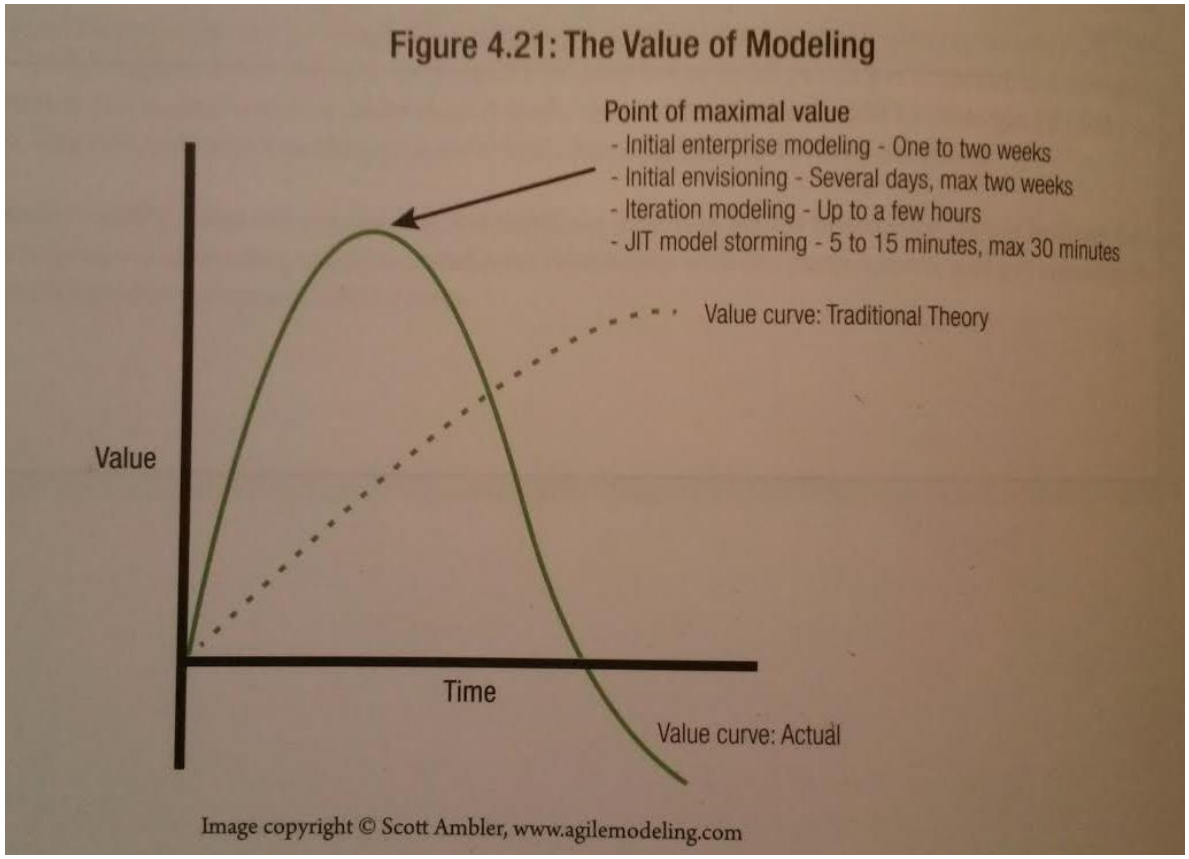
In Continuous Integration Scenarios for ADM, we need easily definable, benchmarkable Quality/SLA indices often based on code size (LOC), such as:

- Code Coverage Percentage of Unit Tests and Integration Tests
- Tangle Index, Duplicate Code and Dead Code percentages
- Technical Debt (number of required days to fix current LOCs)

LOCs as well make project comparison difficult, and they are a poor index of the true "size" of a project. See CISQ standard: http://it-cisq.org/standards/



Why do we need Automation for Sizing?



- The value of modeling (E/R, UML) in Agile is best expressed in the inital phases
- Round-trip engineering of the model is seldom performed (the model does not follow the project)
- ORM becomes the "de facto" on-the-fly model, because *it is part of the code and changes along the project.*

Why do we need Automation for Sizing?

Within a Continuous Integration Environment, it is (practically) impossible to perform model-based roundtrip engineering (every change in the core code should be reproduced back in the original model)

The speed of change makes difficult to use of UML models at logical level for FP counting

If a standard architectural pattern is used (Java Enterprise, Microsoft .NET Architectures), a code-based automatic approach can be used to automate not only testing, but also sizing

The standard architecture is needed to avoid requiring ad-hoc annotations or tools to perform automatic sizing, the model is **already expressed** by current industry standards.

Using the FPA meta-model on ORM

Leverage industry standards for Functional Sizing in Agile Contexts: an example recipe

CPM 4.3.1 in Part 3, Chapter 2 introduces a **stepwise process for establishing the set of logical files** in the application being measured.

This process relies on Data Modeling concepts, and on **Entity-Relationship diagrams** to depict the data:

- Entity
- Associative Entity Type
- Attributive Entity Type
- Entity Subtype
- Relationships (One-to-One, One-to-Many, Many-to-Many)
- Attributes

Using these concepts we can:

- Identify **how many *logical files*** to count (Step 1)
- How to count *RETs (and DETs)* when we identify sub-groups (Step 4)

UML Class Diagram *used as Domain Model (Logical Level)* is considered the best candidate to help identify Logical Files

- Entity -> Class
- Relationship -> Association, Multiplicity and Cardinality
- Generalization -> Subgroup Subtypes
- Composition and Aggregation -> Entity (In)-Dependence

Many proposals attempt to use of other UML diagrams to assist FPA:

- Use Case Diagrams: Use Case -> Elementary Process

However, UML Logical Models **"diverge" quickly** from the real project situation and are **unsuitable** as a base for counting during development and maintenance

Object Relational Mapping is a framework of metadata which map E/R concepts to Classes in an Object-Oriented Paradigm

A well-known ORM standard is JPA 2.0 as specified in [5], but all ORM frameworks are very similar (.NET, RoR, Python…)

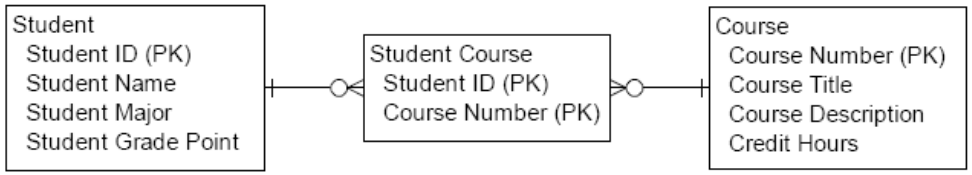
Mapping metadata are expressed as Annotations (AOP) within the definition of relevant Classes

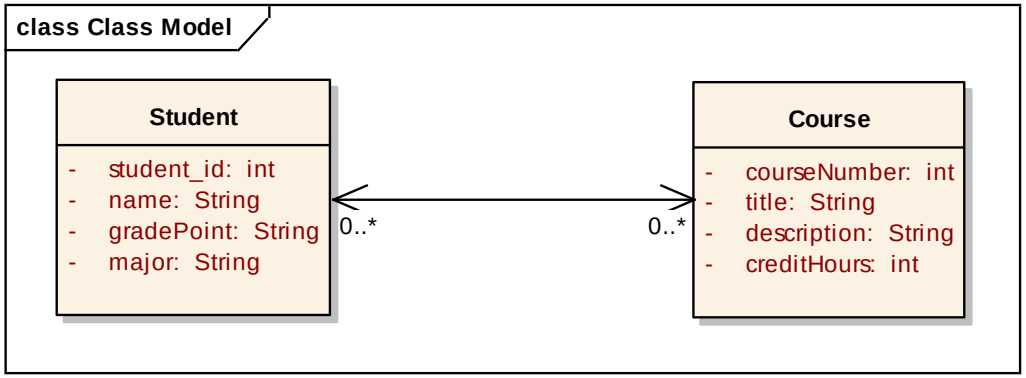
@Entity	Maps a class to a persistent entity
@Id	Maps the PK
@OneToOne	Maps a 1:1 / 1:(1) relationship
@ManyToMany	Maps a N:M / (N):(M) relationship
@ManyToOne	Maps a 1:N / 1:(N) relationship
@ElementCollection	Maps an (Embedded) Collection
@Cascade	Cascades Remove (Persist…) operations on dependent entities

The Advantage of ORM is that the Annotations are used by programmers to code, therefore they evolve the Logical Model while they are coding.

Example 1: Associative Entity

ORM does not use an associative entity and it is therefore clearer for counting correctly





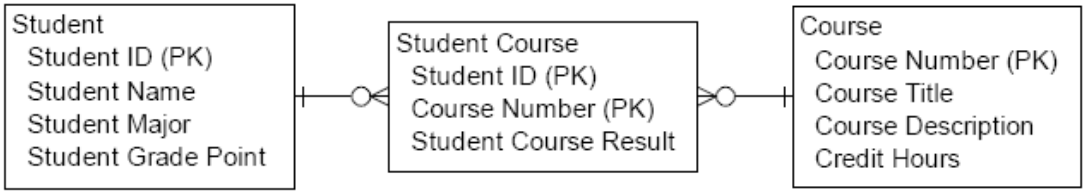

```java
@Entity
public class Student {

        @Id private int student_id;
        String name;
        String gradePoint;
        String major;

        @ManyToMany
        private Collection<Course> courses;


}
```

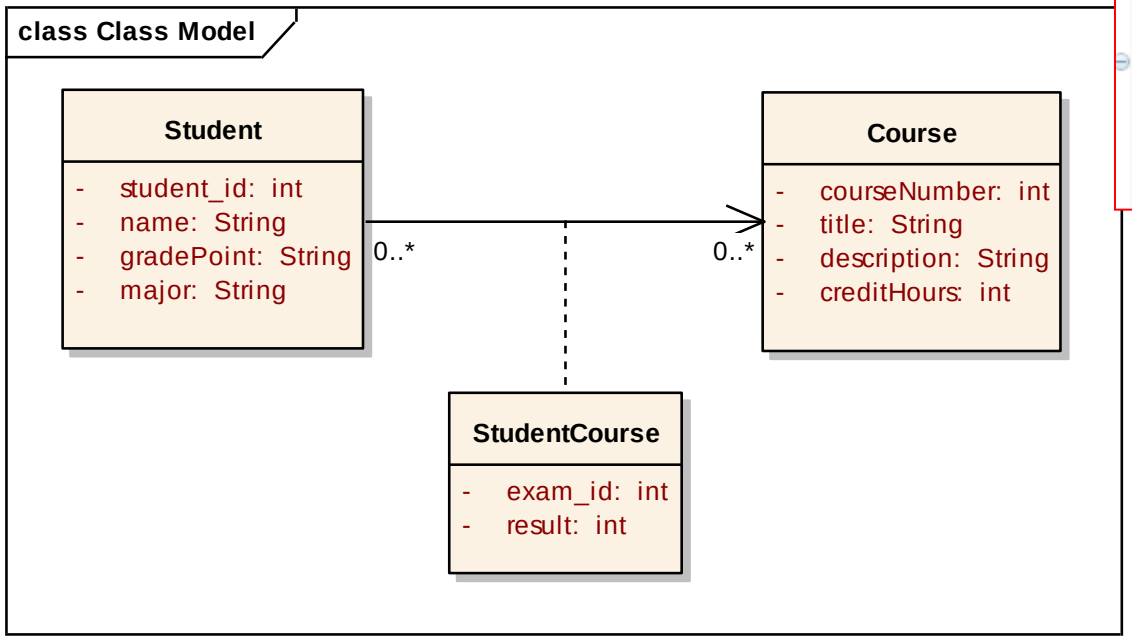
```java
@Entity
public class Course {

    @Id int courseNumber;
    String name;
    int creditHours;

    @ManyToMany(mappedBy="courses")
    private Collection<Student> students;
}
```

Example 2: Associative Entity with RET on one side

The unidirectional association tells us that we are mainly considering the RET from *one* side

This reflects in the ORM mapping as unidirectional






```java
@Entity
public class Student {

        @Id private int student_id;
        String name;
        String gradePoint;
        String major;

        @ManyToMany
        @MapKeyColumn(name="exam_id")
        private HashMap<Integer, Course> exams;

        @ManyToMany
        private Collection<Course> courses;

}
```

```java
@Entity
public class Course {

    @Id int courseNumber;
    String name;
    int creditHours;

}
```

«Summary» Table for ORM

ORM annotations for Entities A,B	When this condition Exists	Then count as LFs with RETs and DETs as follow:
A: @OneToOne B	*Unidirectional* - B is entity dependent (@Cascade: Remove) on A	A: 1 LF, 2 RET, sum DETs; B: -
A: @ManyToMany Collection<B>	*Unidirectional* - B is entity dependent (@Cascade: Remove) on A	A: 1 LF, 2 RET, sum DETs; B: -
A: @ManyToMany Collection<B> B: @ManyToMany Collection<A>	*Bidirectional* - B is entity independent of A	2 LFs, 1 RET and DETs to each
A: @ManyToMany Collection<B> A: @MapKeyColumn Map<Attr,B> B: @ManyToMany Collection<A> B: @ElementCollection Map<Attr,A> *bidirectional*	*Bidirectional* - B is entity independent of A	2 LFs, 2 RET and DETs to each
A: @ManyToMany Collection<B> A: @ElementCollection Map<Attr,B> *unidirectional*	*Unidirectional* - B is entity dependent (@Cascade: Remove) on A	A: 1 LF, 2 RET, sum DETs; B: -
A: @OneToMany Collection<C> B: @OneToMany Collection<C> C: @ManyToOne A C: @ManyToOne B	*Bidirectional* - C is entity independent of A and of B	A: 1 LF, 1 RET, and its DETs; B: 1 LF, 1 RET, and its DETs; C: 1 LF, 1 RET, and its DETs;

Automatically Detecting in a *transparent* way **Elementary Processes** and therefore Counting Transactional Functions (EI / EO / EQ) is an **entirely different matter**

This part of the tool is an ongoing research.

Current early prototype uses Integration Test Code to identify Elementary Processes:

- Theories (Test Fixtures Inputs) and Asserts (Test Output Validations) are used to determine DETs

Another line of development is based on the JSF (Java Server Faces) framework:

- Every @ViewScoped bean is an Elementary Process
- Properties of a @ViewScoped bean are DETs
- Injection points in the @ViewScoped bean are FTRs

A Crude implementation is using CRUD: #EPs = ILF x 4 following ILF's CPX

(A similar approach can be followed in .NET and ROR)

Implementation

Our experiment: an open-source pluggable AFP engine (Data Functions Part)

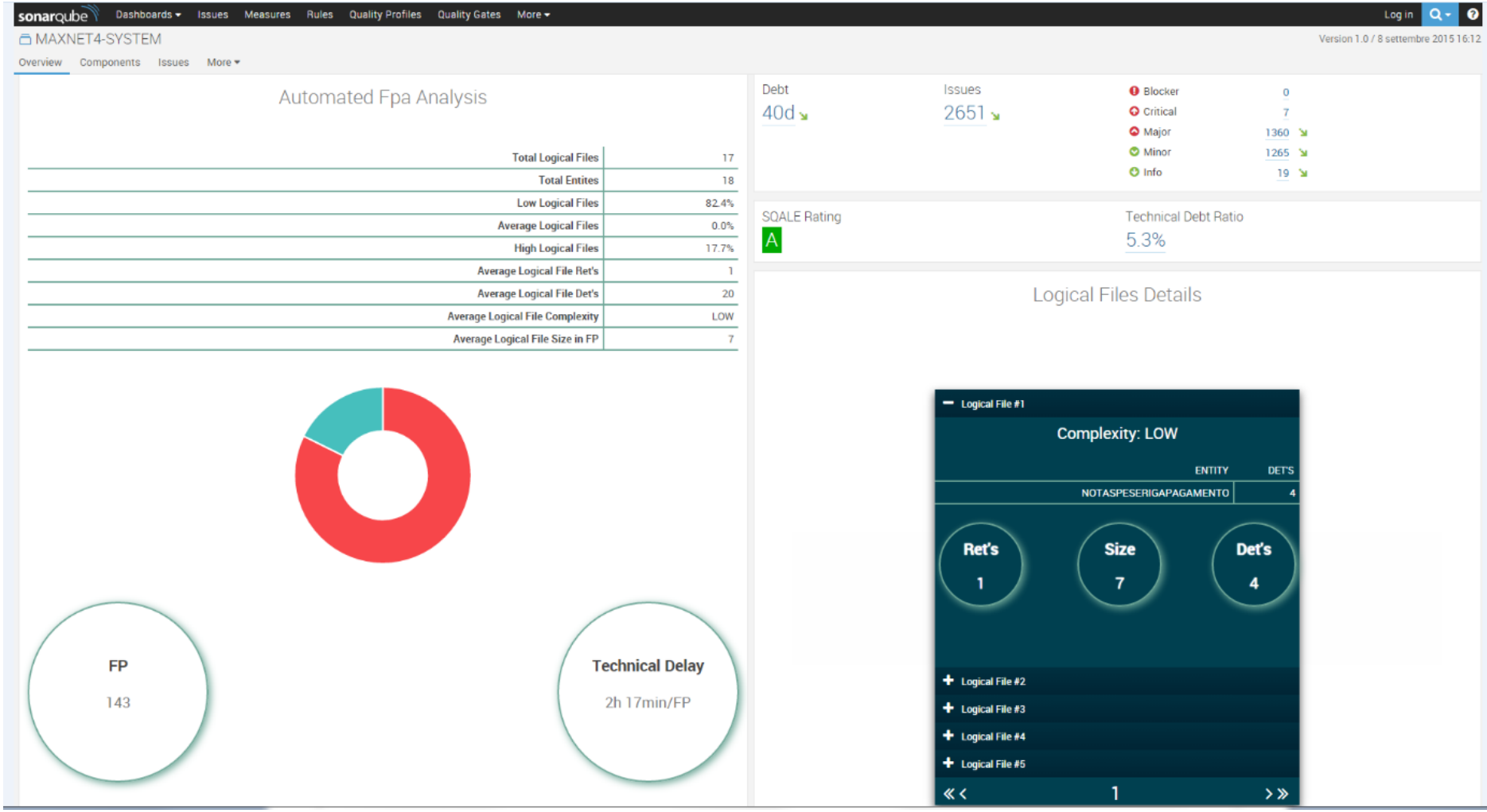
Early Results - Sample Screenshots

Automated Fpa Analysis

Total Logical Files	17
Total Entites	18
Low Logical Files	82.4%
Average Logical Files	0.0%
High Logical Files	17.7%
Average Logical File Ret's	1
Average Logical File Det's	20
Average Logical File Complexity	LOW
Average Logical File Size in FP	7

Agile AFP

Early Results - Sample Screenshots



Conclusions and Future Work

ORM is **simple enough** and **effective enough** as an abstraction for Logical File identification and RET evaluation

We propose some (not all) identification rules **mapped on ORM metadata annotations** (others will follow in future works)

A key for large (and automated) adoption in heterogeneous environments of AFP is a mapping approach ***transparent to programmers*** *(it only requires to apply uniformely the standard technical framework of choice: Java EE, .NET, RoR, etc.)*

The automatic sizing approach can easily **be integrated as a plug-in** in mainstream **open-source** QA technologies such as SonarQUBE, or used as an open-source standalone analyzer

Refine/Debug/Test current implementation, Open-source licensing the project

Engage in Partnerships to collect data to compare AFP result with "true" FP analysis, assess results (contact **thimoty.barbieri@unipv.it**)

Continue research on "automatic" mapping for EI, EO, EQ (based on JSF-standard approach)

Maintain SonarQube plug-in adding EI/EO/EQ counter

Bibliography

[1] Albrecht A.J., «Measuring Application Development Productivity», *IBM Applications Development Symposium*, Monterey, CA, 1979

[2] IFPUG, «Function Points Counting Practices Manual», version 4.3.1, International Function Point User Group, 2006

[3] Cantone G., Pace D., Calavaro G., «Applying Function Point to Unified Modeling Language: Conversion Model and Pilot Study» in Proceedings of 10° Intl. Symposium of software metrics (IEEE METRICS 2004), Chicago (IL), September 11-17, 2004, pp. 209-291, IEEE CS Press, 2004

[4] Lavazza L., Agostini A. «Automated Measurement of UML Models: an open toolset approach», Journal of Object Technology, May/June 2005

[5] De Michiel, L. «JSR-000317 Java™ Persistence 2.0», http://jcp.org/aboutJava/communityprocess/final/jsr317/, 2009

[6] CISQ: Using Software Measurement in SLAs: Integrating CISQ Size and Structural Quality Measures into Contractual Relationships: http://it-cisq.org/wp-content/uploads/2015/07/Using-Software-Measurement-in-SLAs-Integrating-CISQ-Size-and-Structural-Quality-Measures-into-Contractual-Relationships.pdf