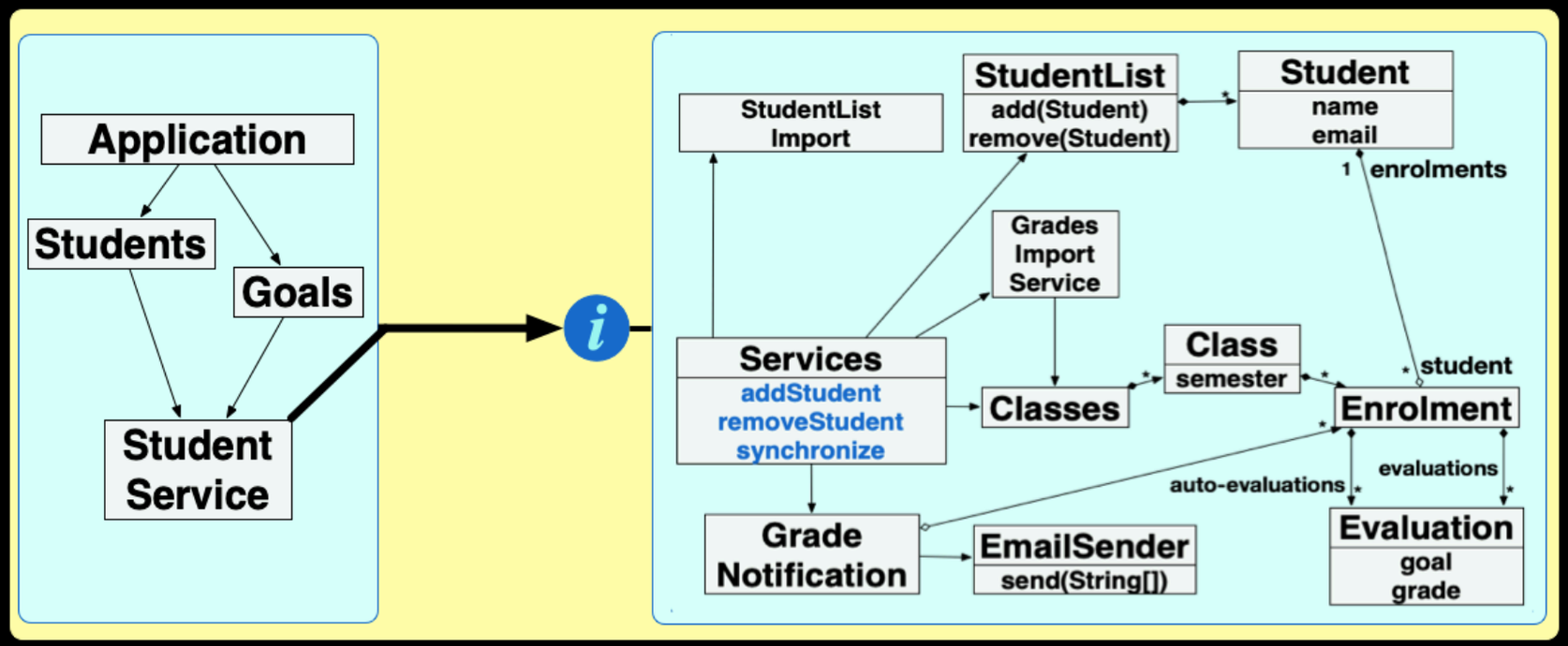
Merge and Code Review

Paulo Borba
Informatics Center
Federal University of Pernambuco

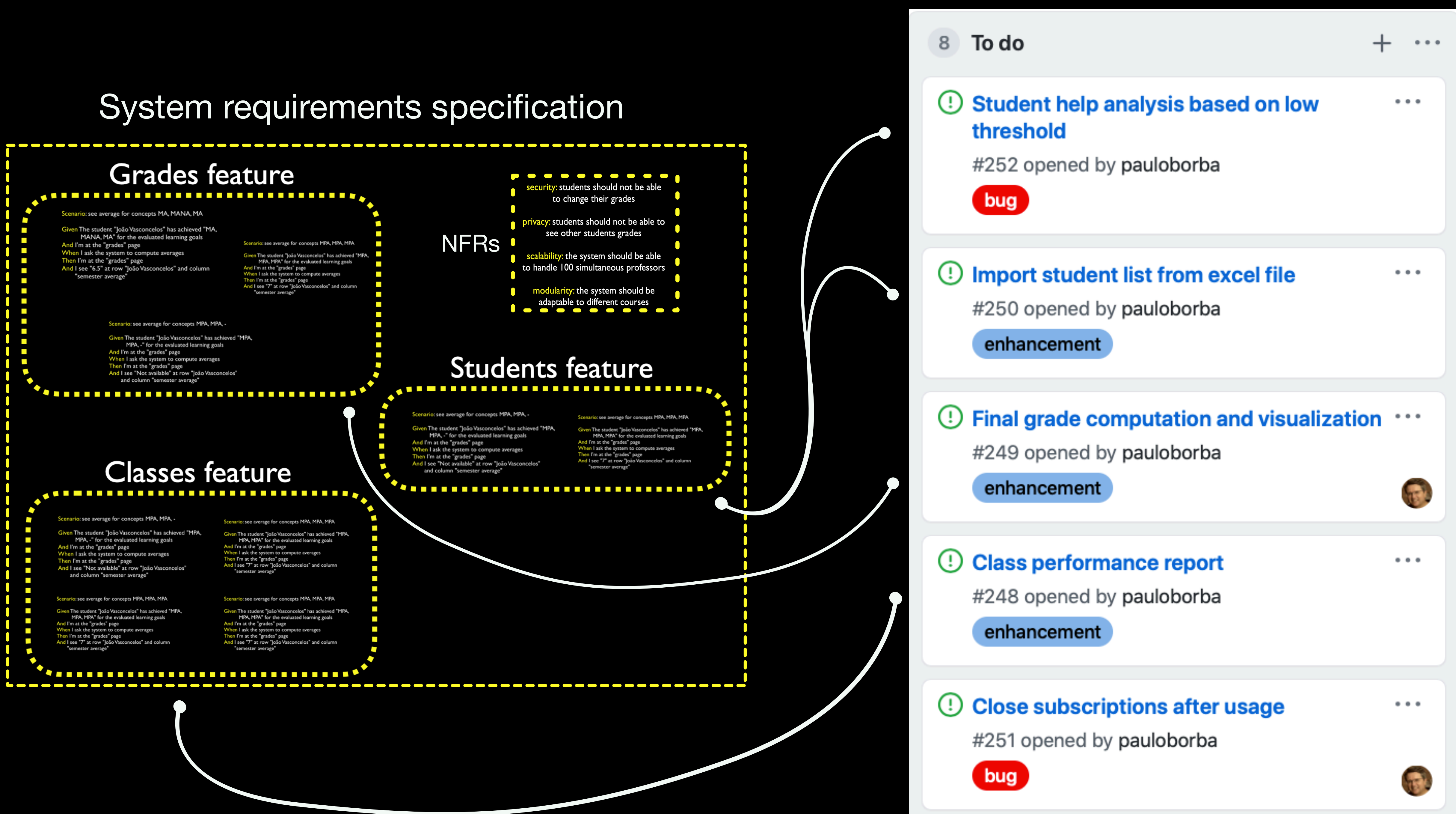
pauloborba.cin.ufpe.br

Context

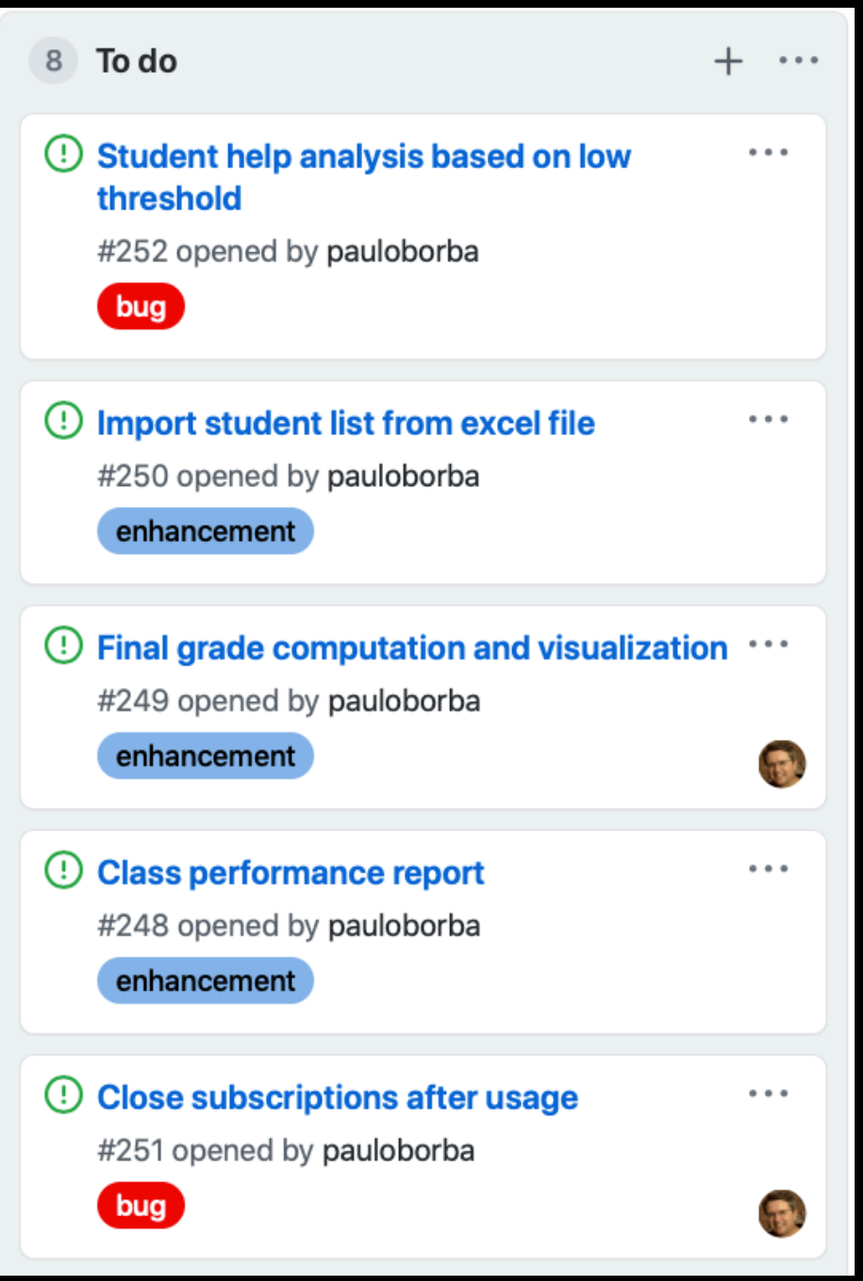
Collaborative software development

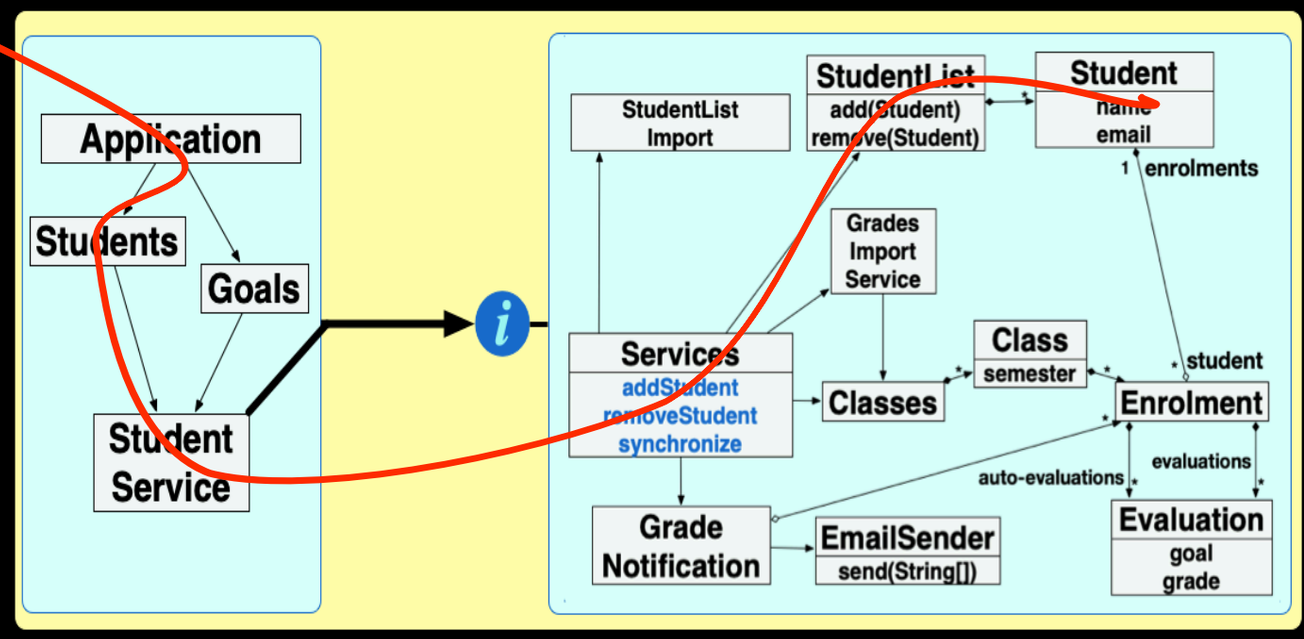


Task structure is often derived from requirements structure

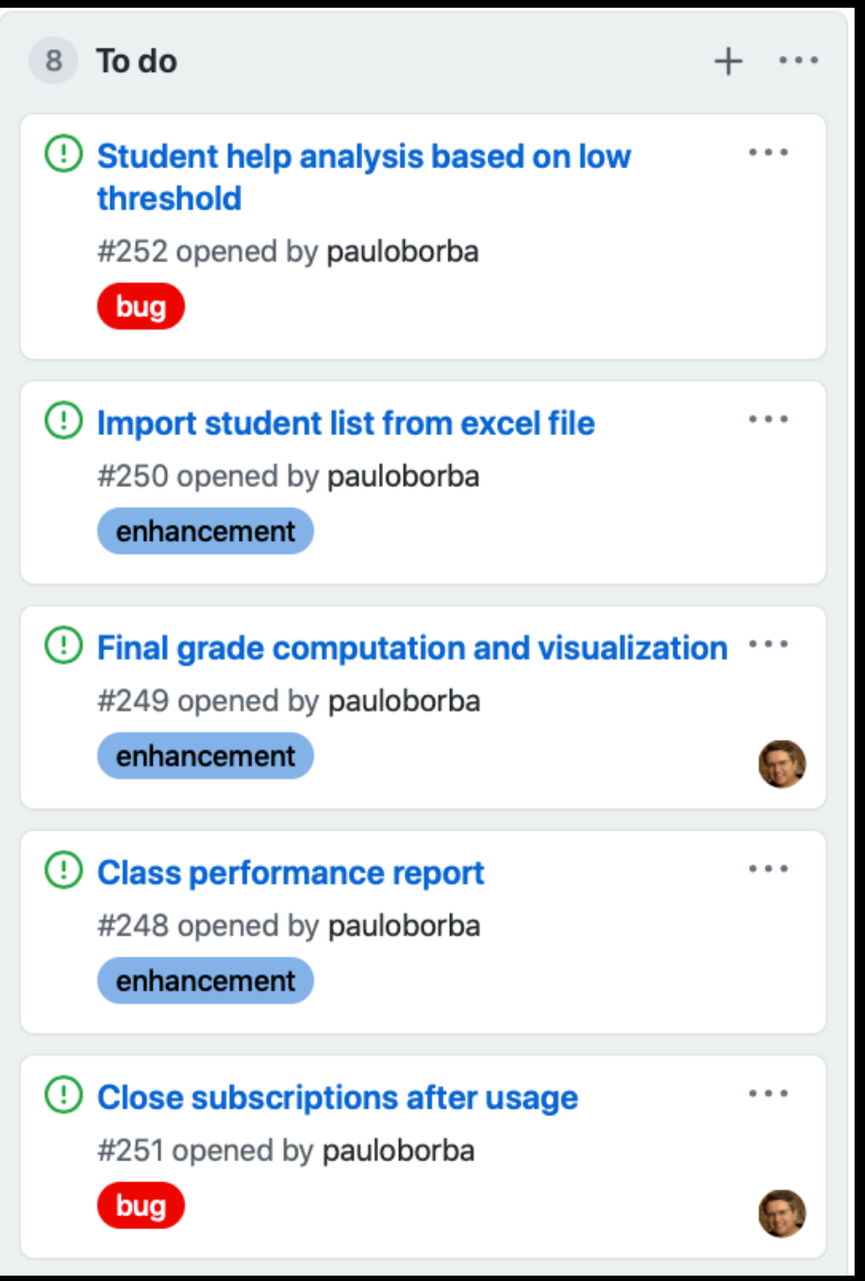


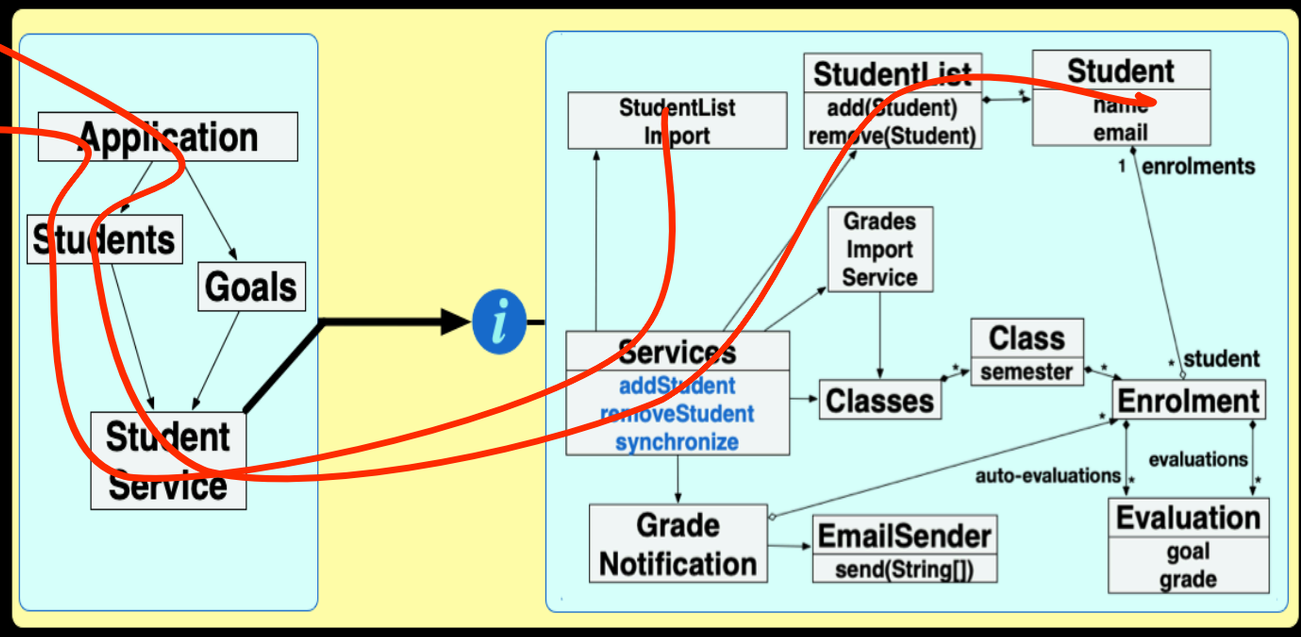
Tasks are often crosscutting



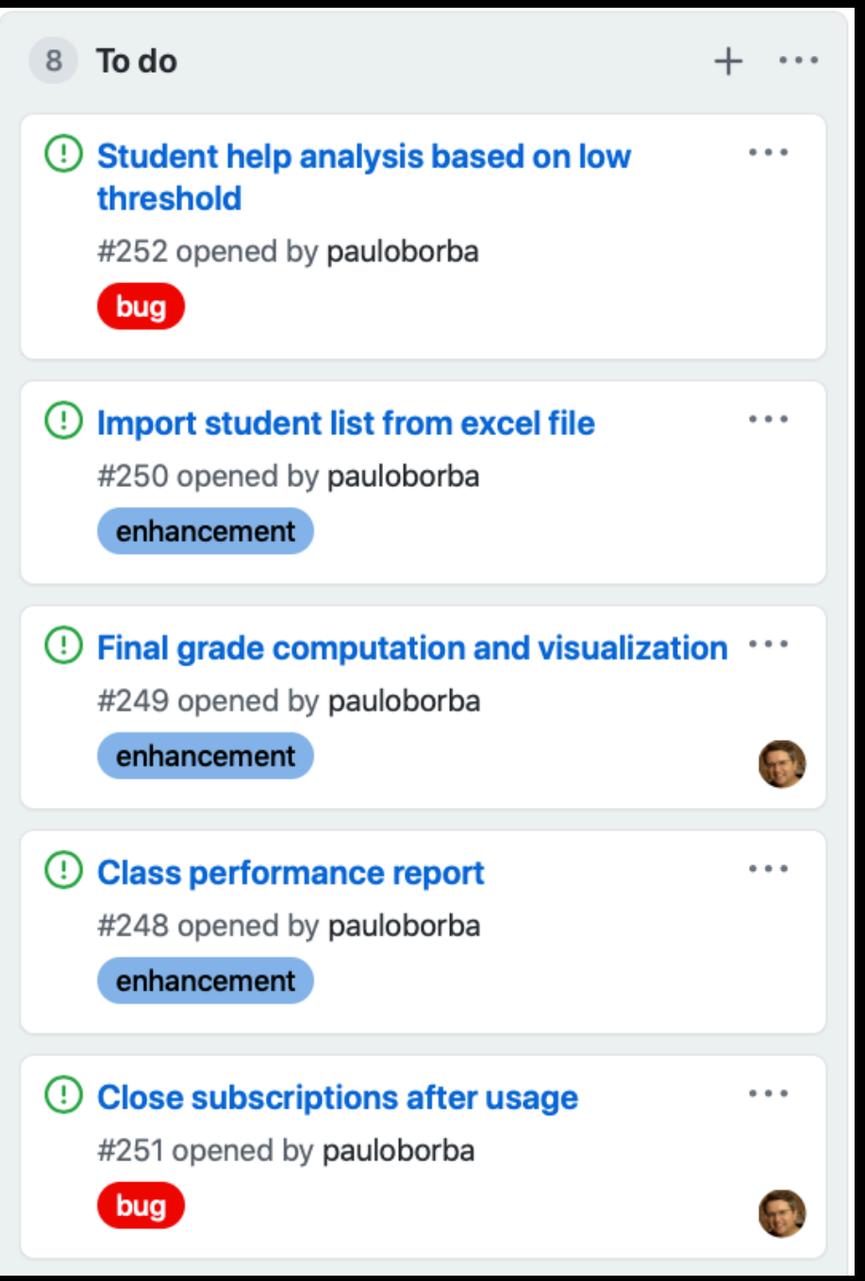


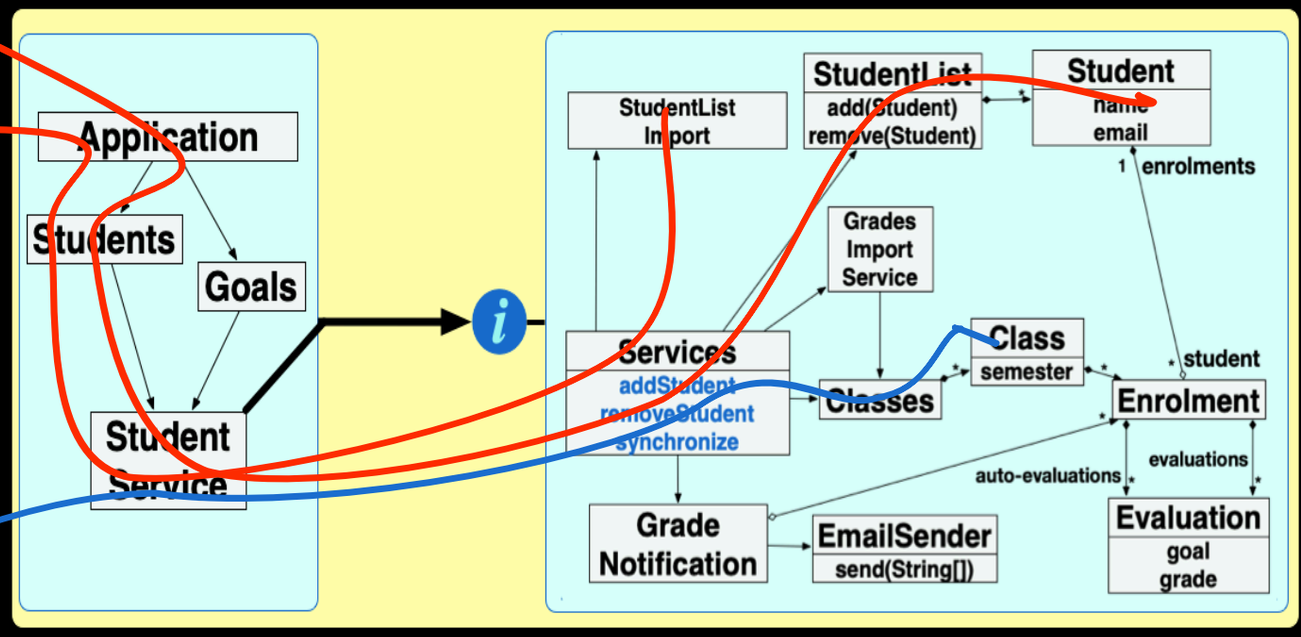
Tasks might involve changing classes in common



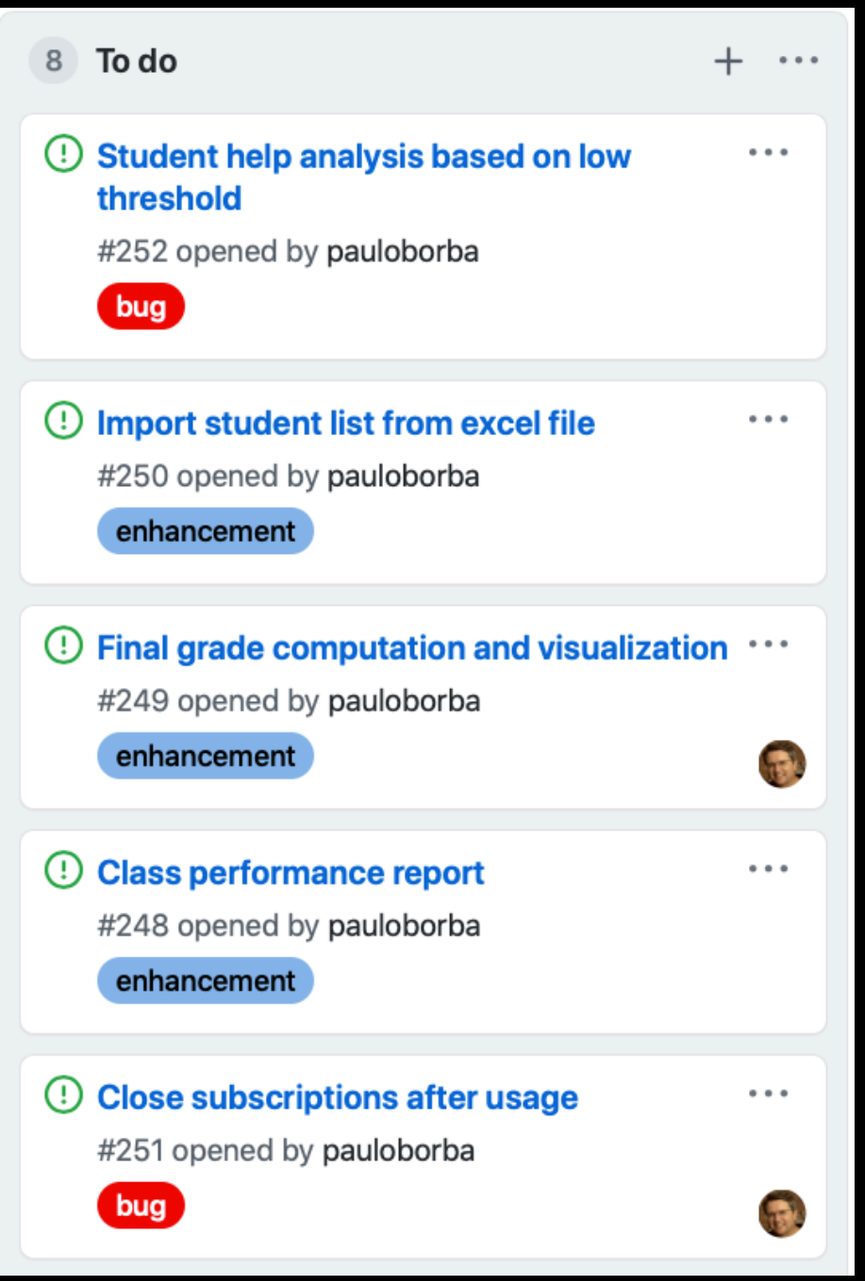


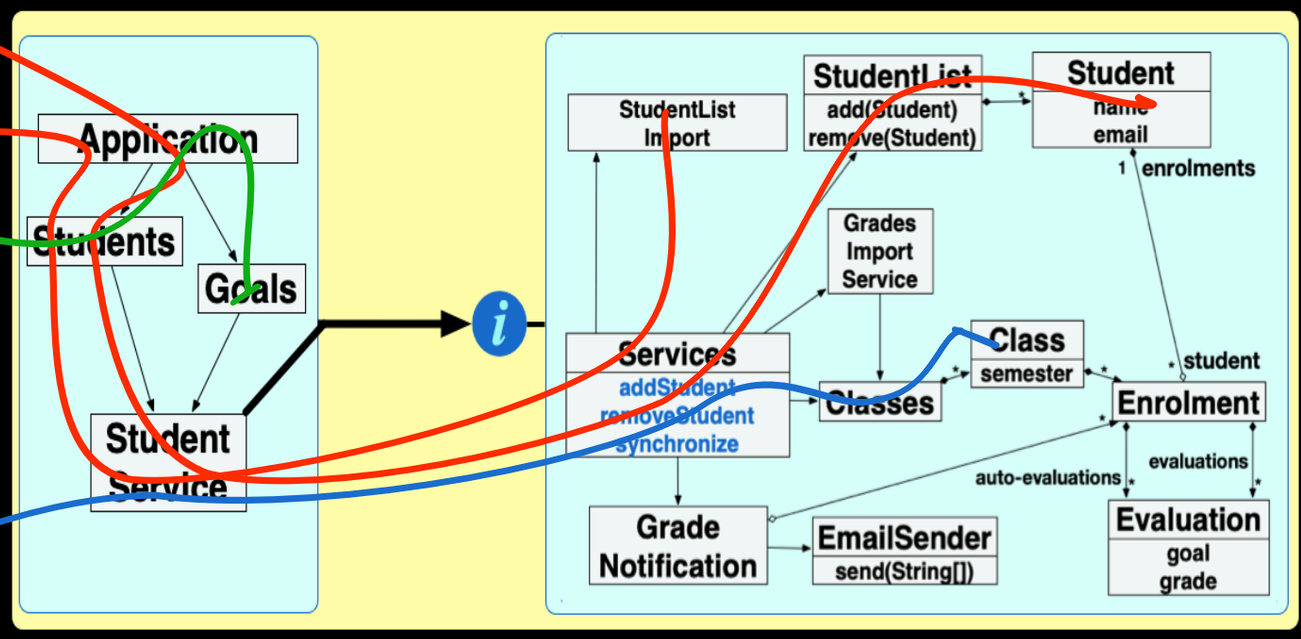
Task structure often does not match code structure



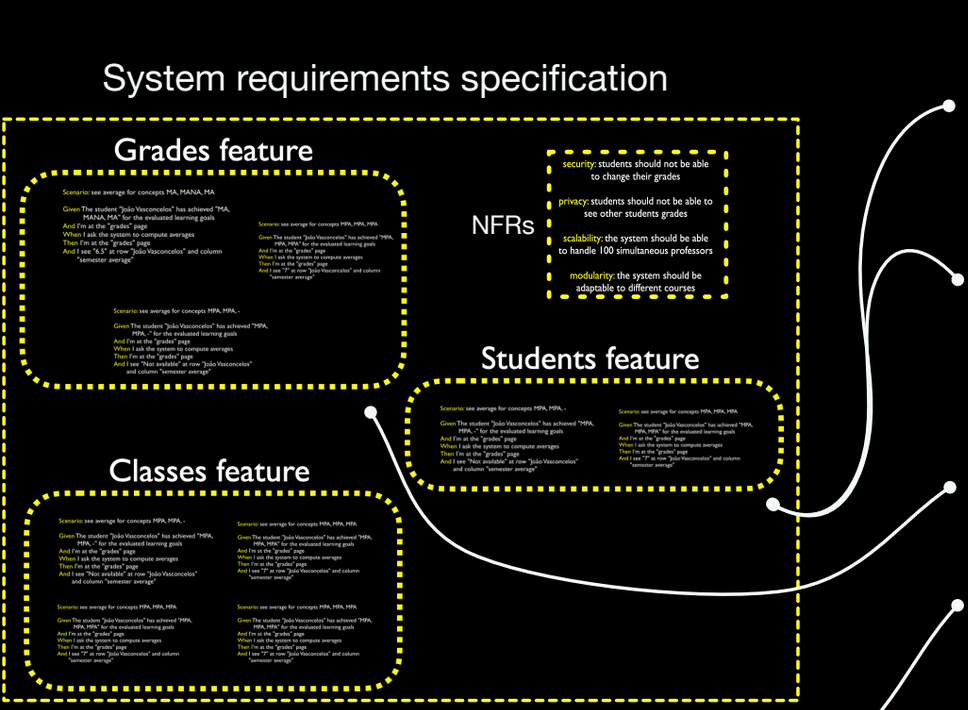


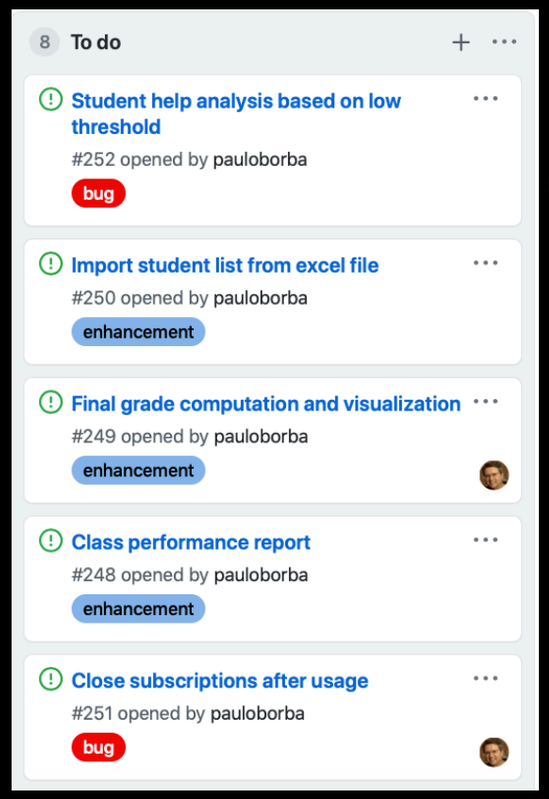
Modular development is not always possible, no matter the investment in modularity

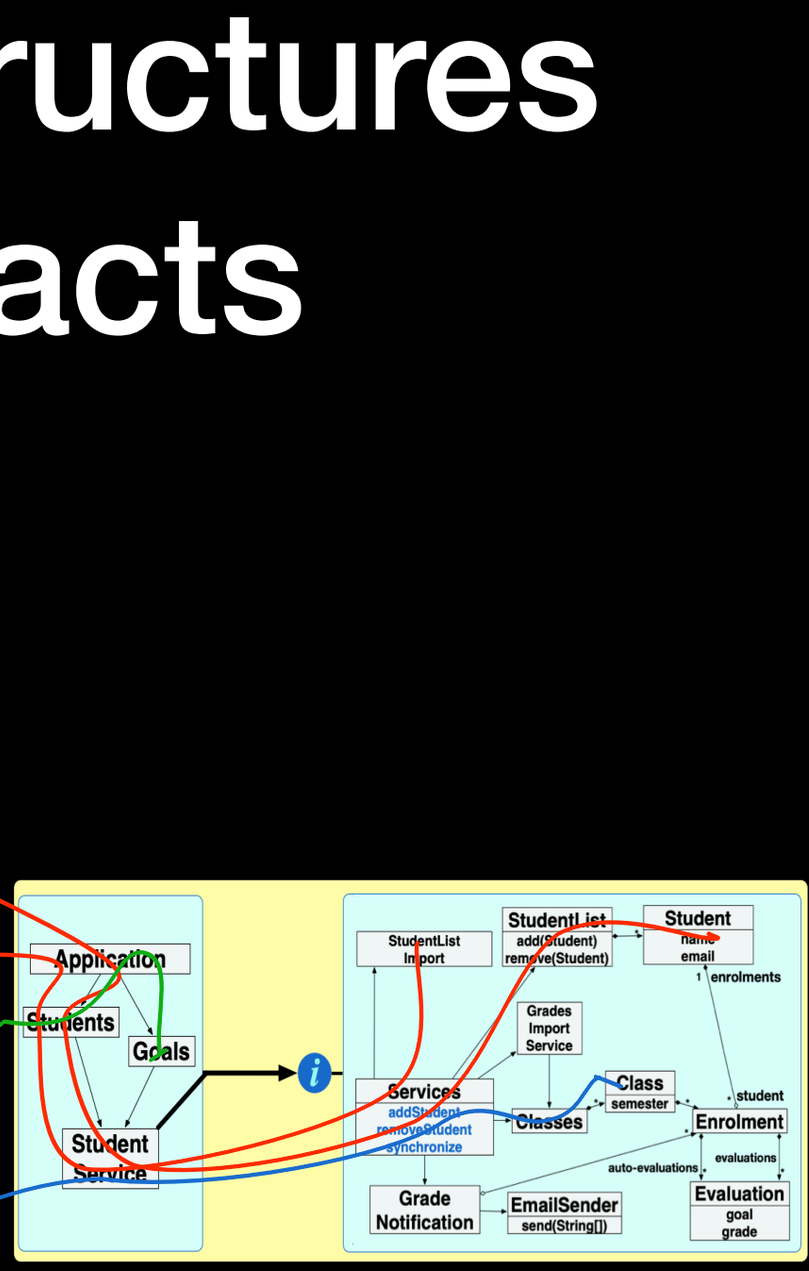




Different modular structures for different artifacts



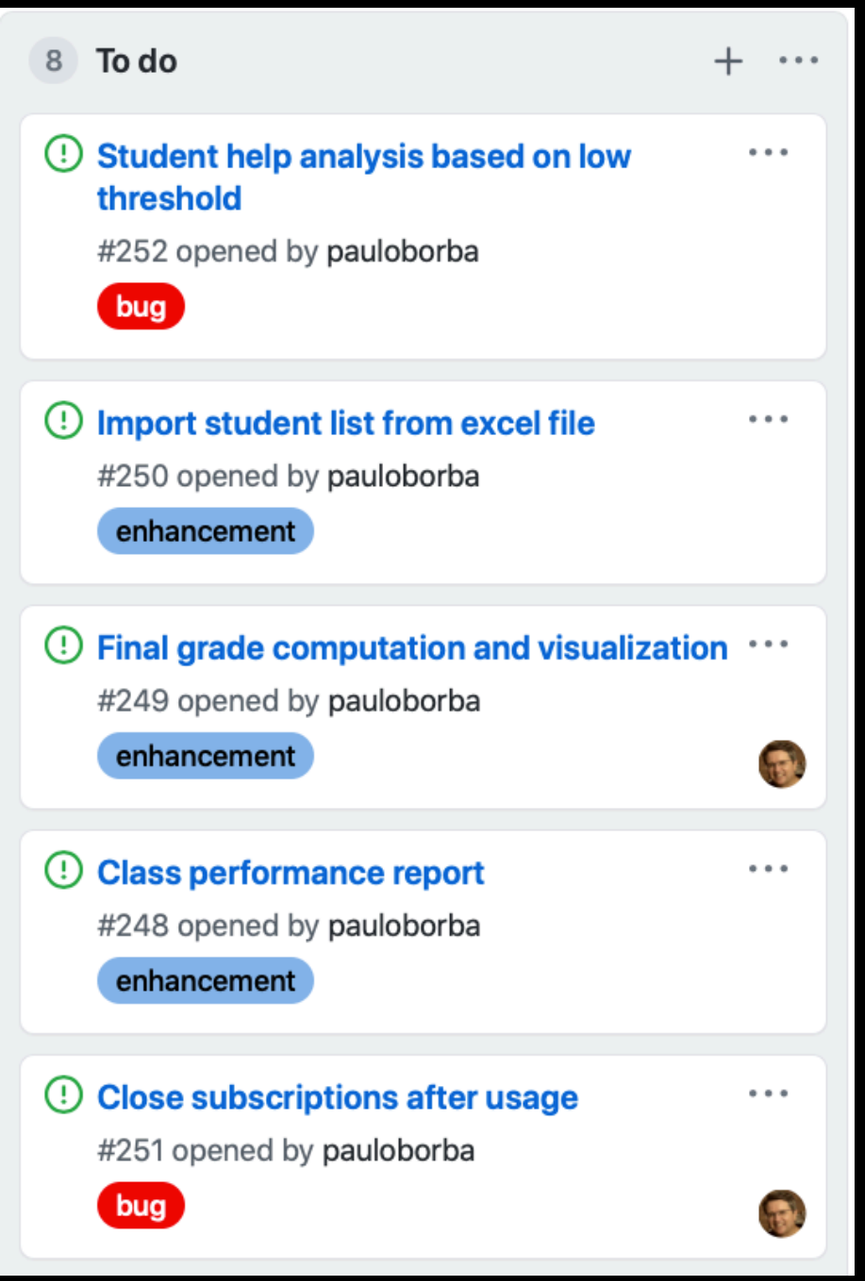


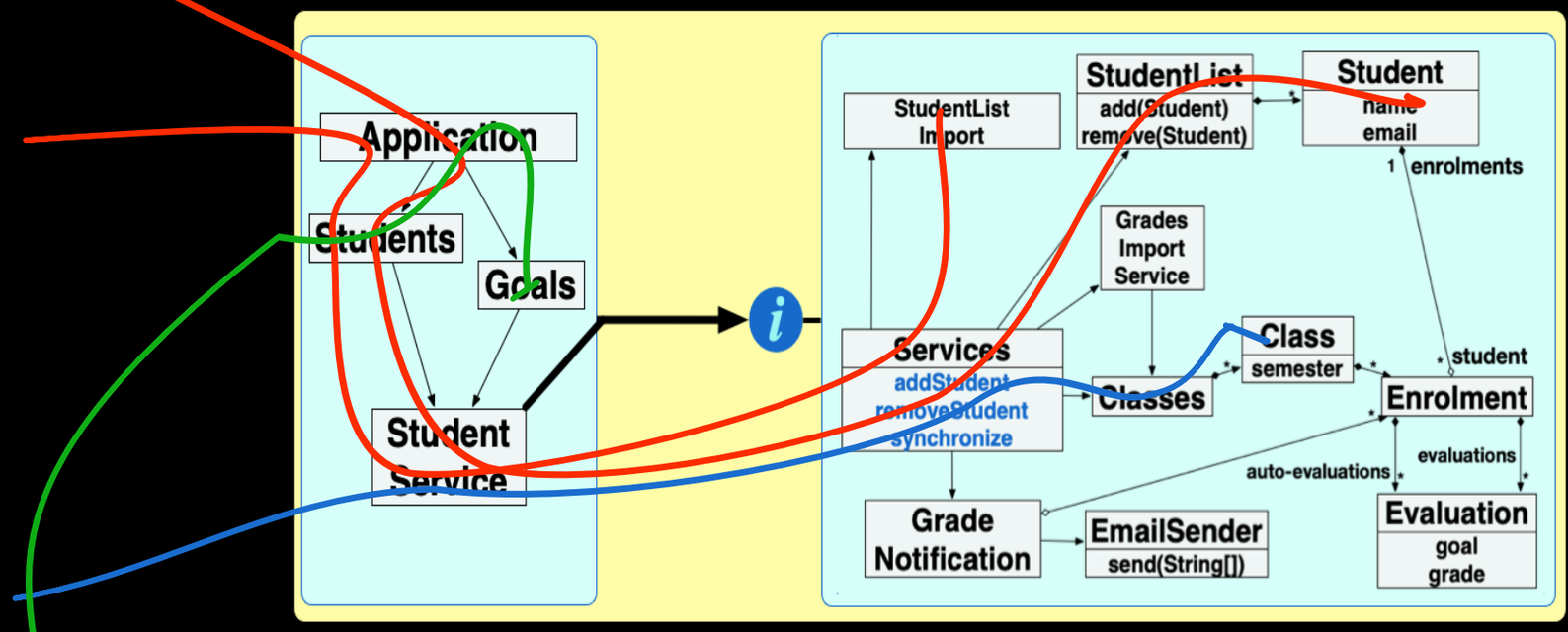


Problem

What could go wrong during code integration?

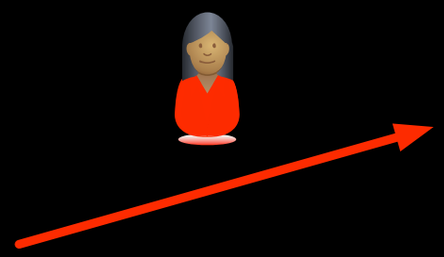
(assuming branching and merging is available)





Merge conflicts

(textual conflicts)



```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
    removeDuplicateWords();
  }
}
```

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
  }
}
```

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
    normalizeWhitespace();
  }
}
```

merge

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
    <<<<<<
    removeDuplicateWords();
    ======
    normalizeWhitespace();
    >>>>>>
  }
}
```

occur in many merge scenarios [Kasi&Sarma, Brun et al, Zimmermann]

affect productivity and quality [Meyer et al]

Avoiding merge conflicts at any cost…

- rushing to finish changes first
- partial check-ins
- continuous integration
- trunk-based development
- feature toggles

only partially motivated by the need to avoid merge conflicts, but with drawbacks anyway

Similar situation with MVC

packages

app
controllers
articles_controller.rb
posts_controller.rb
models
article.rb
post.rb
views
article
posts
_table.html.erb
archived.html.erb
edit.html.erb
index.html.erb
show.html.erb

post slice

module structure established by the kinds of classes

module structure established by features and tasks

Solution

How to reconcile different modular structures with collaborative parallel development?

The code integration process should be simpler and more reliable

Developers waste time by manually resolving conflicts that could be automatically solved

```
class Text {
  public String text;
  …
  void removeDuplicateWords() {…}
  void cleanText() {
    removeComments();
  }
}
```

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
  }
}
```

merge

```
class Text {
  public String text;
  …
  void normalizeWhitespace() {…}
  void cleanText() {
    removeComments();
  }
}
```

```
class Text {
  public String text;
  …
<<<<<<
void removeDuplicateWords() {…}
======
void normalizeWhitespace() {…}
>>>>>>
  void cleanText() {
    removeComments();
  }
}
```

false positive of unstructured merge

Current merge tools might integrate conflicting changes without warning developers

```
class Text {
  public String text;
  …
  boolean equals(…) {…}
  void cleanText() {
    removeComments();
  }
}
```

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
  }
}
```

merge

```
class Text {
  public String text;
  …
  void cleanText() {
    removeComments();
  }
  boolean equals(…) {… …}
}
```

```
class Text {
  public String text;
  …
  boolean equals(…) {…}
  void cleanText() {
    removeComments();
  }
  boolean equals(…) {… …}
}
```

false negative of unstructured merge

Substantial evidence about conflicts…

- Merge conflicts: 7-19% [Kasi&Sarma], 6-42% [Brun et al], 23-46% [Zimmermann]
- Build conflicts: 2-15% [Kasi&Sarma], 0.1-10% [Brun et al]
- Test conflits: 6-35% [Kasi&Sarma], 3-28% [Brun et al]

Course details

8/9	Apresentação do curso
10/9	Gerência de configuração 1, operações básicas
15/9	Gerência de configuração 2, operações avançadas e integração contínua
17/9	Merge textual (diff3)
22/9	Prática
24/9	CBSoft
29/9	Análise e transformação de programas (TreeSitter) e Merge semiestruturado (s3m)
1/10	Prática
6/10	Merge estruturado (Mergiraf, LastMerge, sepmerge)
8/10	Prática
13/10	Merge semântico (SAM, SMS)
15/10	Merge semântico (SAM, SMS)
20/10	Análise estática de código (Soot, INFER)
22/10	Prática
27/10	Prática
29/10	Análise dinâmica de código (Jalangi)
3/11	Prática
5/11	Geração de testes de unidade (LLMs, Evosuite, Randoop)
10/11	Prática
12/11	Revisão de código
17/11	Prática
19/11	Uso de LLMs para merge e revisão de código
24/11	Projeto
26/11	Projeto
1/12	Projeto
3/12	Projeto
10/12	Apresentação do projeto
15/12	Apresentação do projeto

Evaluation items

- Project (7)
- Exercise sets (3)
- Class participation (extra 0.5)

Merge and Code Review

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br