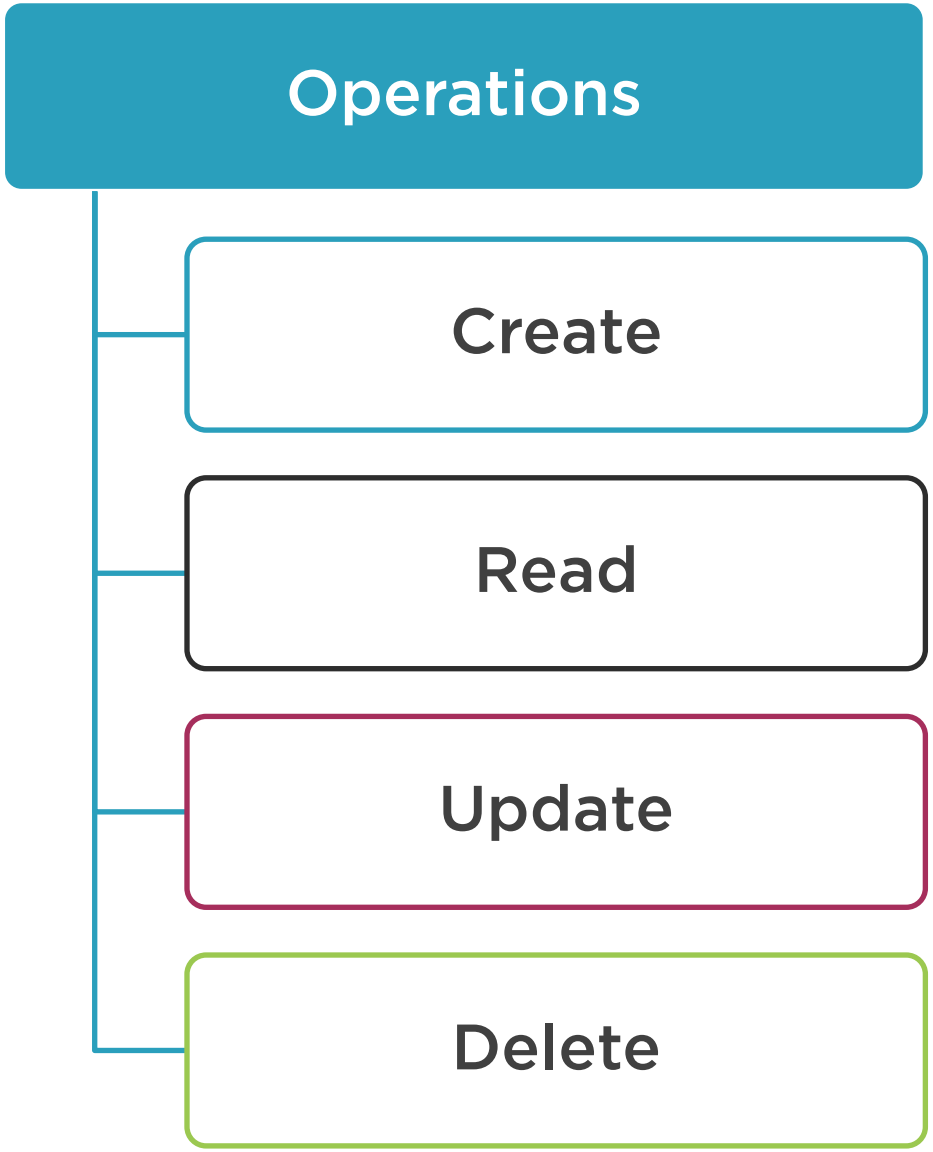
Refactoring Towards a Task-based Interface

Vladimir Khorikov

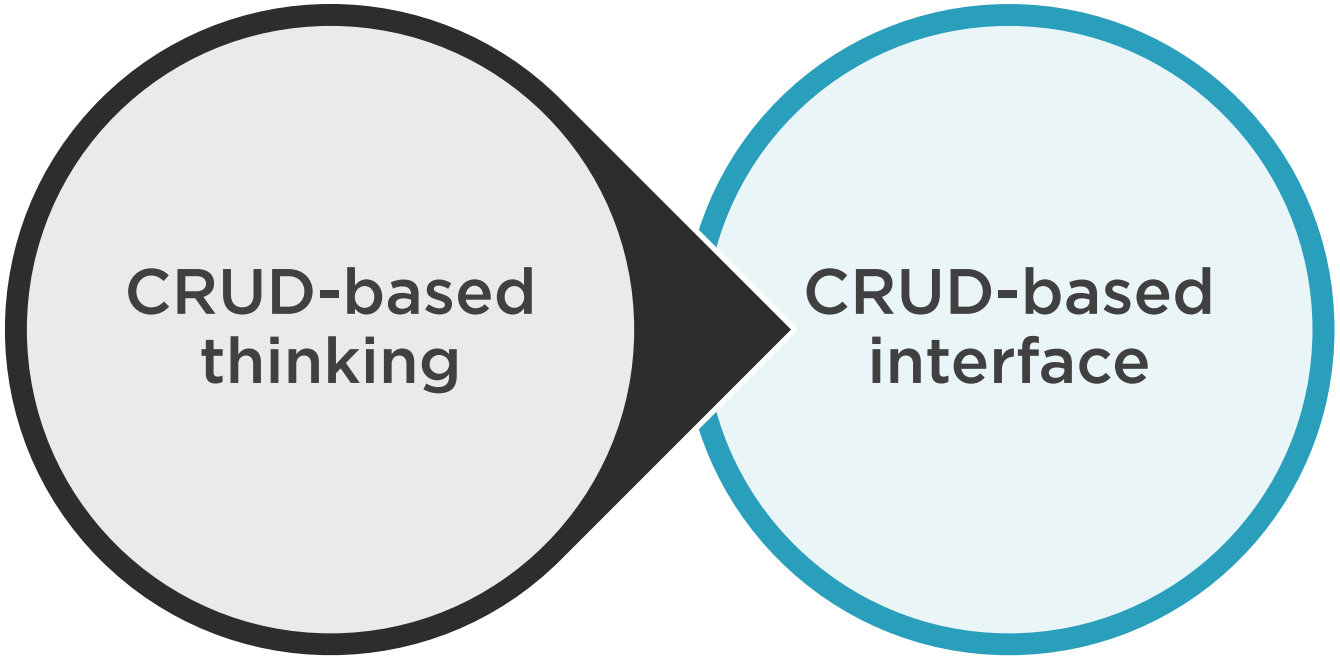
@vkhorikov www.enterprisecraftsmanship.com

CRUD-based Interface



- Not the best way to structure the application
- Can have a devastating effect on your system

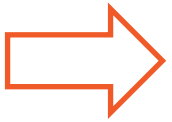
CRUD-based Interface



Problems with CRUD-based Interface

Growth of complexity

Too many features in a single method

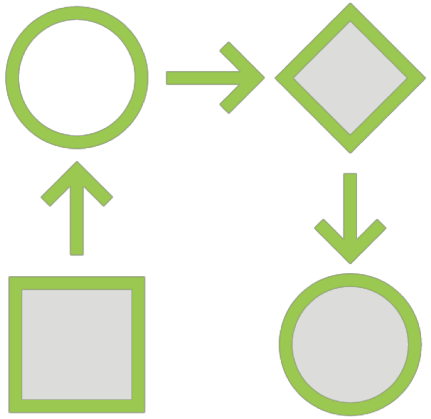


Increase in number of bugs

Problems with CRUD-based Interface

Experts' view

≠

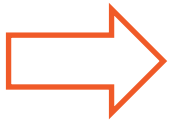


Implementation

- ✗ Experts don't speak in CRUD terms
- ✗ Lack of ubiquitous language

Problems with CRUD-based Interface

Lack of unified language



Maintainability issues

Have to translate from experts' language

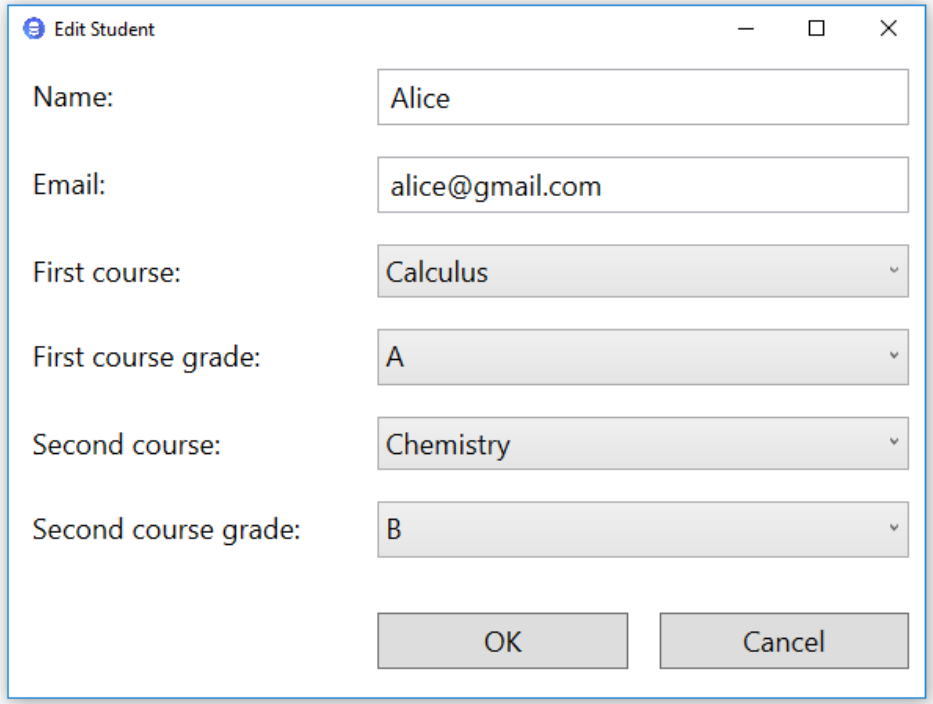
Reduces capacity to understand the task

Problems with CRUD-based Interface

Damaging the user experience

The user has to investigate the interface on their own

Problems with CRUD-based Interface



Editing personal information

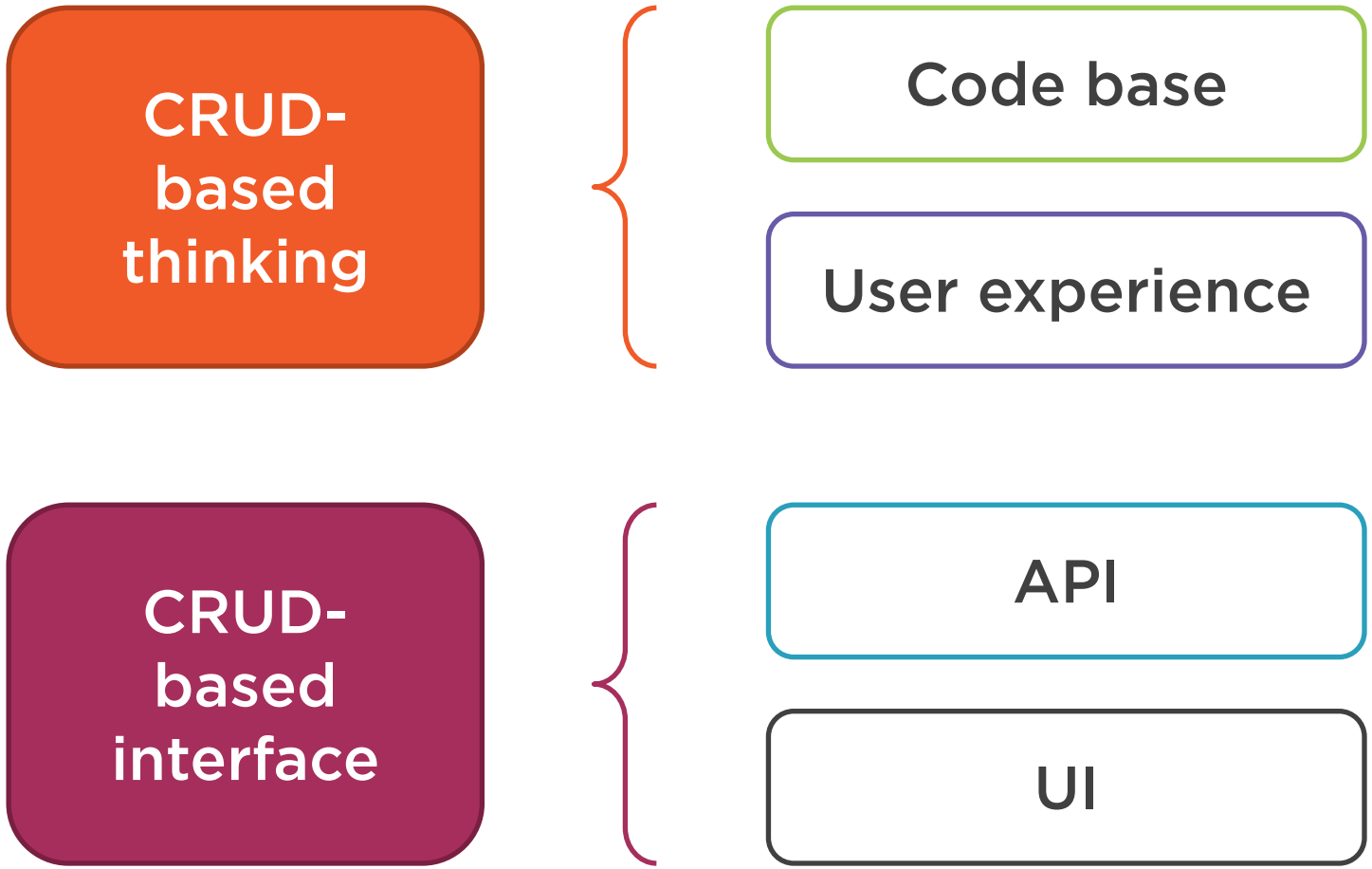
Enrolling into a new course

Transferring to another course

Disenrolling from a course

Hard to build a proper mental model

Problems with CRUD-based Interface



Why is CRUD-based interface so widely spread?

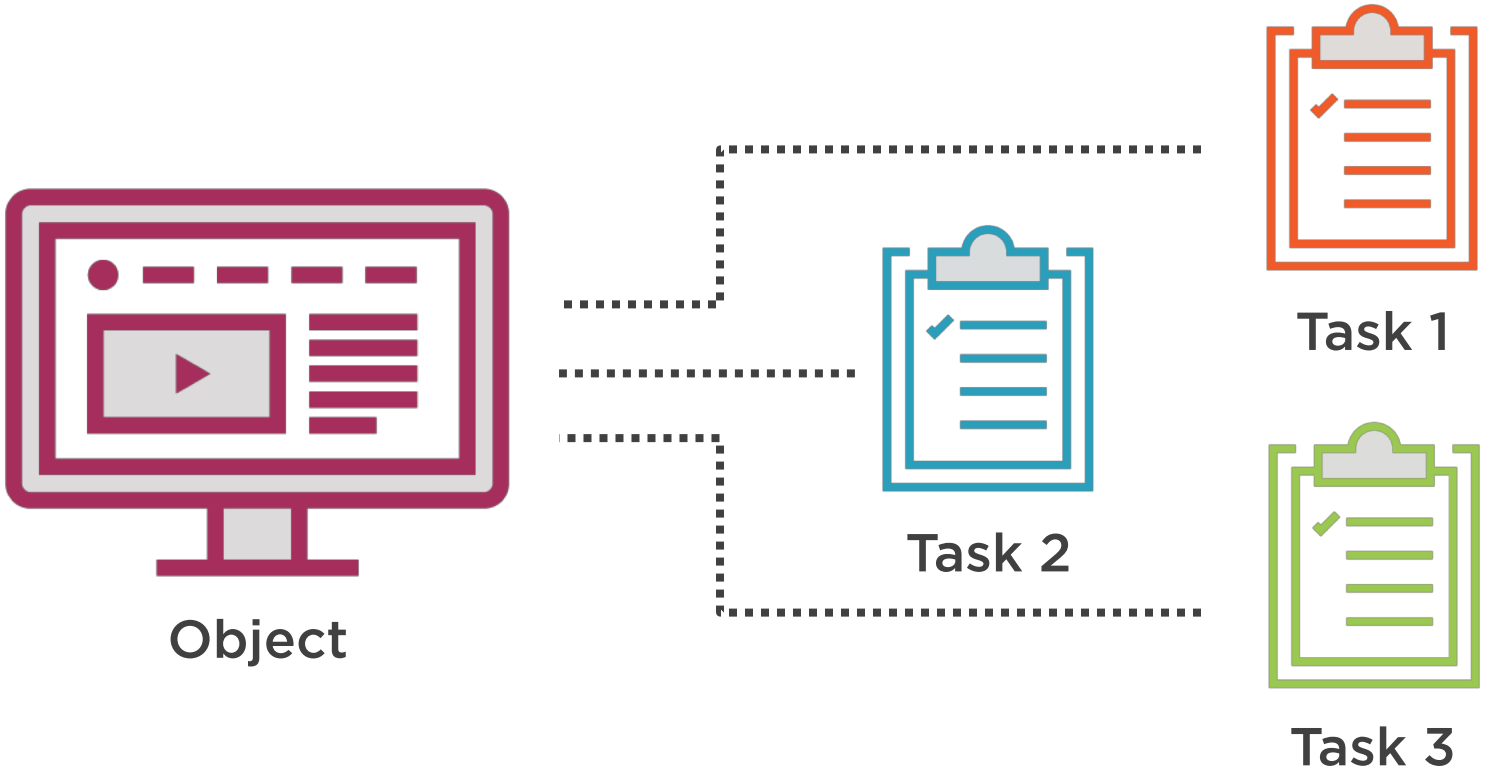
CRUD-based Interface

OOP : Everything is an object

CRUD-based thinking : Everything is CRUD

Task-based interface

Task-based Interface



Separate window

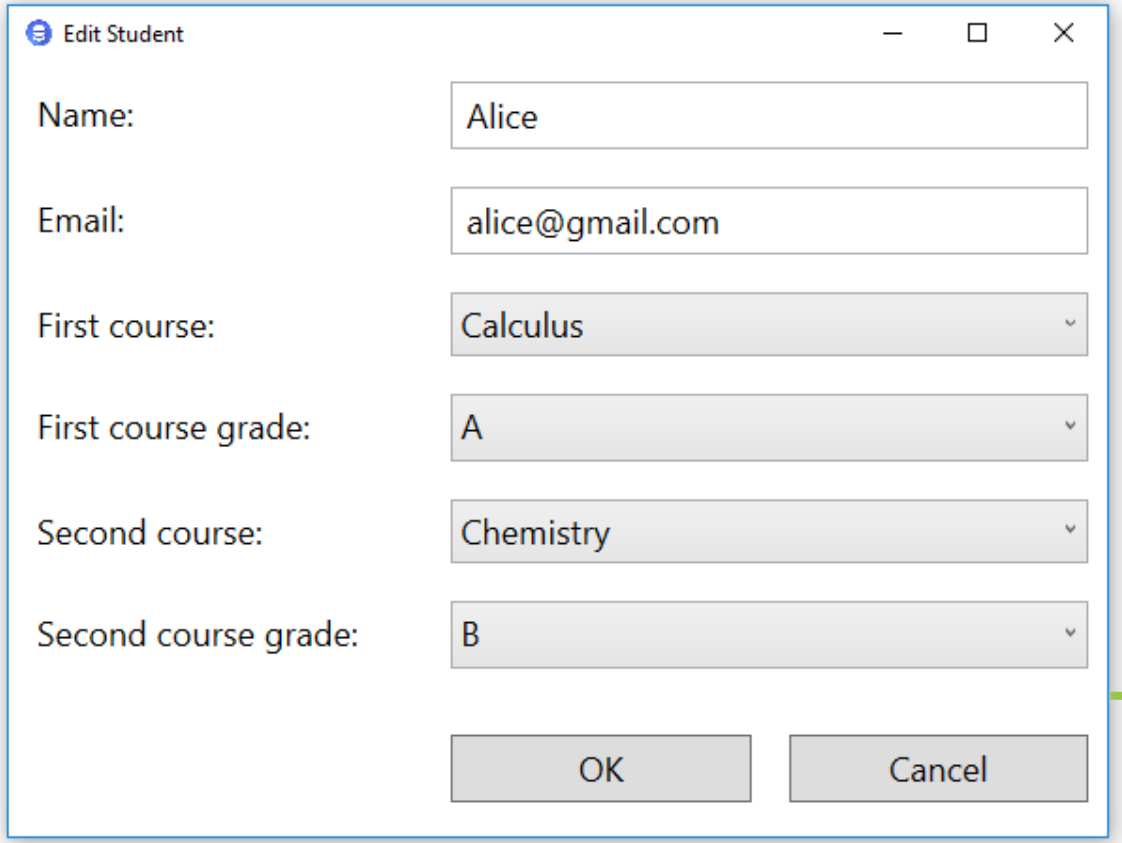
Separate API

Task-based Interface

Intuitive UI → Each window does one thing

- Restore the single responsibility principle
- Code base simplification
- Improves user experience

Untangling the Update Method



Editing personal info

Enrolling into a course

Transferring

Disenrolling from a course

Object-centric

Task-centric

Recap: Untangling the Update Method

- ✓ **Untangled the student update method**
  - Editing personal information
  - Enrolling into a new course
  - Transferring to another course
  - Disenrolling from a course
- ✓ **Simplified the code base**

Recap: Untangling the Update Method

```
public IActionResult Disenroll(long id, int enrollmentNumber,
    [FromBody] StudentDisenrollmentDto dto)
{
    Student student = _studentRepository.GetById(id);
    if (student == null)
        return Error($"No student found for Id {id}");

    if (string.IsNullOrWhiteSpace(dto.Comment))
        return Error("Disenrollment comment is required");

    Enrollment enrollment = student.GetEnrollment(enrollmentNumber);
    if (enrollment == null)
        return Error($"No enrollment found with number '{enrollmentNumber}'");

    student.RemoveEnrollment(enrollment, dto.Comment);
    _unitOfWork.Commit();

    return Ok();
}
```

```
if (HasEnrollmentChanged(dto.Course1, dto.Course1Grade, firstEnrollment))
{
    if (string.IsNullOrWhiteSpace(dto.Course1))
    {
        if (string.IsNullOrWhiteSpace(dto.Course1DisenrollmentComment))
            return Error("Disenrollment comment is required");

        Enrollment enrollment = firstEnrollment;
        student.RemoveEnrollment(enrollment);
        student.AddDisenrollmentComment(enrollment, dto.Course1DisenrollmentComment);
    }

    if (string.IsNullOrWhiteSpace(dto.Course1Grade))
        return Error("Grade is required");

    Course course = _courseRepository.GetByName(dto.Course1);

    if (firstEnrollment == null)
    {
        student.Enroll(course, Enum.Parse<Grade>(dto.Course1Grade));
    }
    else
    {
        firstEnrollment.Update(course, Enum.Parse<Grade>(dto.Course1Grade));
    }
}
```

Cyclomatic complexity

```
public IActionResult Disenroll(long id, int enrollmentNumber,
    [FromBody] StudentDisenrollmentDto dto)
{
    Student student = _studentRepository.GetById(id);
    if (student == null)
        return Error($"No student found for Id {id}");

    if (string.IsNullOrWhiteSpace(dto.Comment))
        return Error("Disenrollment comment is required");

    Enrollment enrollment = student.GetEnrollment(enrollmentNumber);
    if (enrollment == null)
        return Error($"No enrollment found with number '{enrollmentNumber}'");

    student.RemoveEnrollment(enrollment, dto.Comment);
    _unitOfWork.Commit();

    return Ok();
}
```

Cyclomatic complexity

Recap: Untangling the Update Method

Validation

Domain model

Commitment

```
public IActionResult Disenroll(long id, int enrollmentNumber,
    [FromBody] StudentDisenrollmentDto dto)
{
    Student student = _studentRepository.GetById(id);
    if (student == null)
        return Error($"No student found for Id {id}");

    if (string.IsNullOrWhiteSpace(dto.Comment))
        return Error("Disenrollment comment is required");

    Enrollment enrollment = student.GetEnrollment(enrollmentNumber);
    if (enrollment == null)
        return Error($"No enrollment found with number '{enrollmentNumber}'");

    student.RemoveEnrollment(enrollment, dto.Comment);

    _unitOfWork.Commit();

    return Ok();
}
```

Recap: Untangling the Update Method

```
public class StudentDto
{
    long Id { }
    string Name { }
    string Email { }

    string Course1 { }
    string Course1Grade { }
    string Course1DisenrollmentComment { }
    int? Course1Credits { }

    string Course2 { }
    string Course2Grade { }
    string Course2DisenrollmentComment { }
    int? Course2Credits { }
}
```

```
public class StudentEnrollmentDto
{
    string Course { }
    string Grade { }
}

public class StudentDisenrollmentDto
{
    string Comment { }
}

public class StudentPersonalInfoDto
{
    string Name { }
    string Email { }
}
```

Unnecessary fields

No redundancies

Avoid DTOs that are full of “holes”.

Recap: Untangling the Update Method

Simplicity is inherent for task-based interface

Tasks provide right level of granularity and intent

Dealing with Create and Delete Methods

But what about create and delete API endpoints?

Dealing with Create and Delete Methods

Create student → Register student

Delete student → Unregister student

Operations named after CRUD terms

Task-based Interface

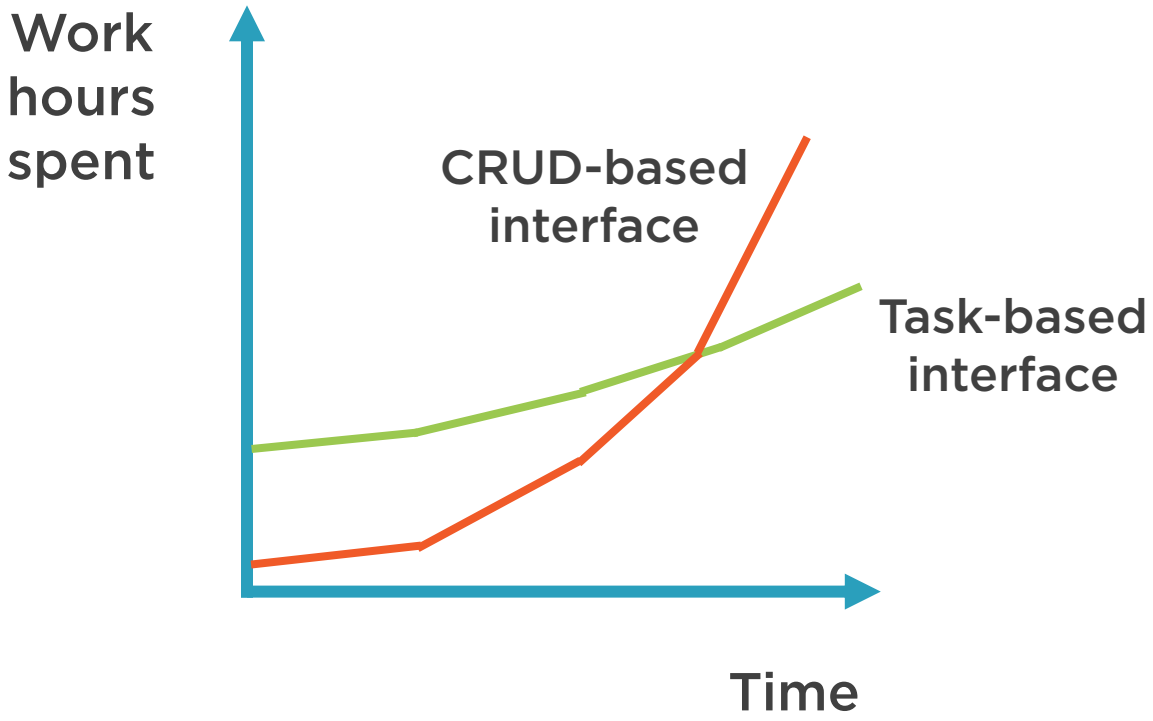
Task-based Interface ≠ CQRS

- Can have one with or without the other
- CQRS often goes hand in hand with CQRS

Task-based Interface

Sometimes, CRUD-based interface is just fine

Task-based Interface



Summary

Refactored towards task-based interface

CRUD-based thinking: fitting all operations into CRUD terminology

CRUD-based interface: result of CRUD-based thinking

- "Interface" means both UI and API

Problems with CRUD-based interface

- Uncontrolled growth of complexity
- Lack of ubiquitous language
- Damaging the user experience

Task-based interface

- Result of identifying each task the user can accomplish with an object
- Affects both UI and API

Summary

Avoid CRUD-based language

- A sign of CRUD-based thinking
- Create & Delete -> Register & Unregister

Avoid DTOs with “holes” in them

- Example: DisenrollmentComment and Grade in StudentDto

In the Next Module

Segregating Commands and Queries