Safely Evolving Legacy Code

Slides up at scottsauber.com

scottsauber

Thank you to our Sponsors!

dev up

SYLLOGISTEKS

TEAM Daugherty Est. 1985

ArchitectNow

MiTek®

TURNBERRY SOLUTIONS

SecureIdeas professionally evil®

Microsoft Azure

MySQL™

ADAPTIVE™ SOLUTIONS GROUP

HUNTER Engineering Company

kepler "Your time, spent wisely."

LOCKBASE

Audience

- Poll
    - How many people maintain legacy code?
    - How many people enjoy it?

Agenda

- What is Legacy Code
- Options for making changes to Legacy Code
- Process for safely doing so
- How I’ve evolved legacy code with real world applications

Goals

- Give you ideas on how you can safely work with legacy code today
- This is **not** about how to completely rewrite your app, this is about **evolving**

Who am I?

- Director of Engineering at Lean TECHniques
- Microsoft MVP
- Co-organizer of Iowa .NET User Group
- Redgate Community Ambassador
- Dometrain Author
- Blog at scottsauber.com

“Legacy Code is code without tests.”

Michael Feathers

“Legacy Code is valuable code we’re afraid to change.”

JB Rainsberger

Characteristics of Legacy Code

- Hard to understand
- Changing code often breaks the app in unexpected ways
- Long cycle times to add or change functionality
- Huge estimates for seemingly small changes
- Stories seemingly drag on across days, weeks, months
- 3rd party brought in for assessment

A JIRA issue gets created…

…for “that app”

3 options when changing legacy code

- Edit and Pray
- The Great Rewrite To Solve All Our Problems™
- Cover and Modify

Question to ask yourself:
How much time do you have?

Answer 1:

I need to add this new functionality today!

I need to add new functionality today!

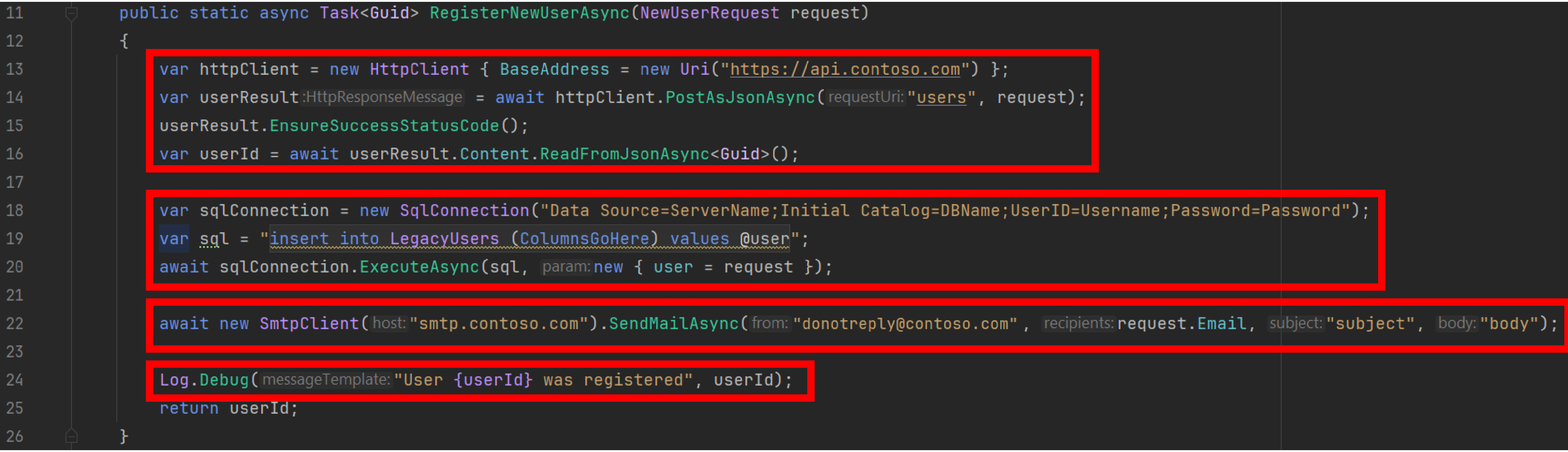
- Adding “just a few more lines” to the god classes/functions is not an option
- Find seams in existing code
- We **<u>should not</u>** refactor without existing tests in place
- Make minimal changes to untested code with one of two techniques
- Sprout
- Wrap

Sprout

Sprout

- Create new method/function/class to house new behavior
- TDD the new behavior
- Insert new, tested code into existing code (existing code untested)
- In the case of .NET you can call new C# code from old VB code

Sprout Example – Existing Code



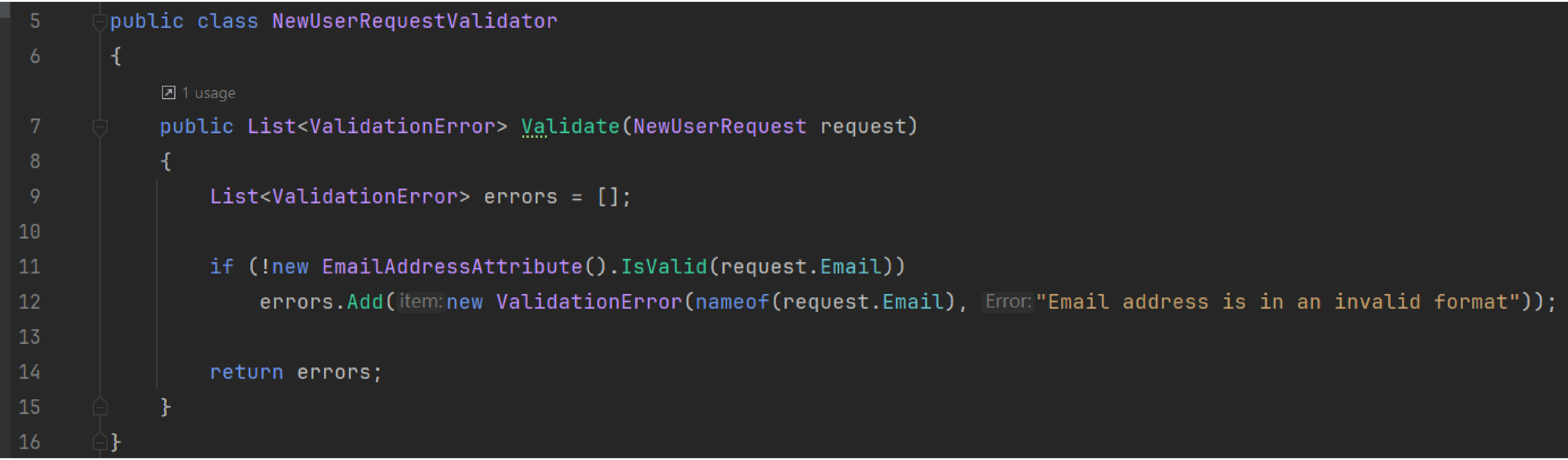
```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    var httpClient = new HttpClient { BaseAddress = new Uri("https://api.contoso.com") };
    var userResult :HttpResponseMessage = await httpClient.PostAsJsonAsync( requestUri: "users", request);
    userResult.EnsureSuccessStatusCode();
    var userId = await userResult.Content.ReadFromJsonAsync<Guid>();

    var sqlConnection = new SqlConnection("Data Source=ServerName;Initial Catalog=DBName;UserID=Username;Password=Password");
    var sql = "insert into LegacyUsers (ColumnsGoHere) values @user";
    await sqlConnection.ExecuteAsync(sql,  param: new { user = request });

    await new SmtpClient( host: "smtp.contoso.com").SendMailAsync( from: "donotreply@contoso.com" ,  recipients: request.Email,  subject: "subject",  body: "body");

    Log.Debug( messageTemplate: "User {userId} was registered", userId);
    return userId;
}
```

scottsauber

New Requirement!

Do not allow an invalid email address to be registered

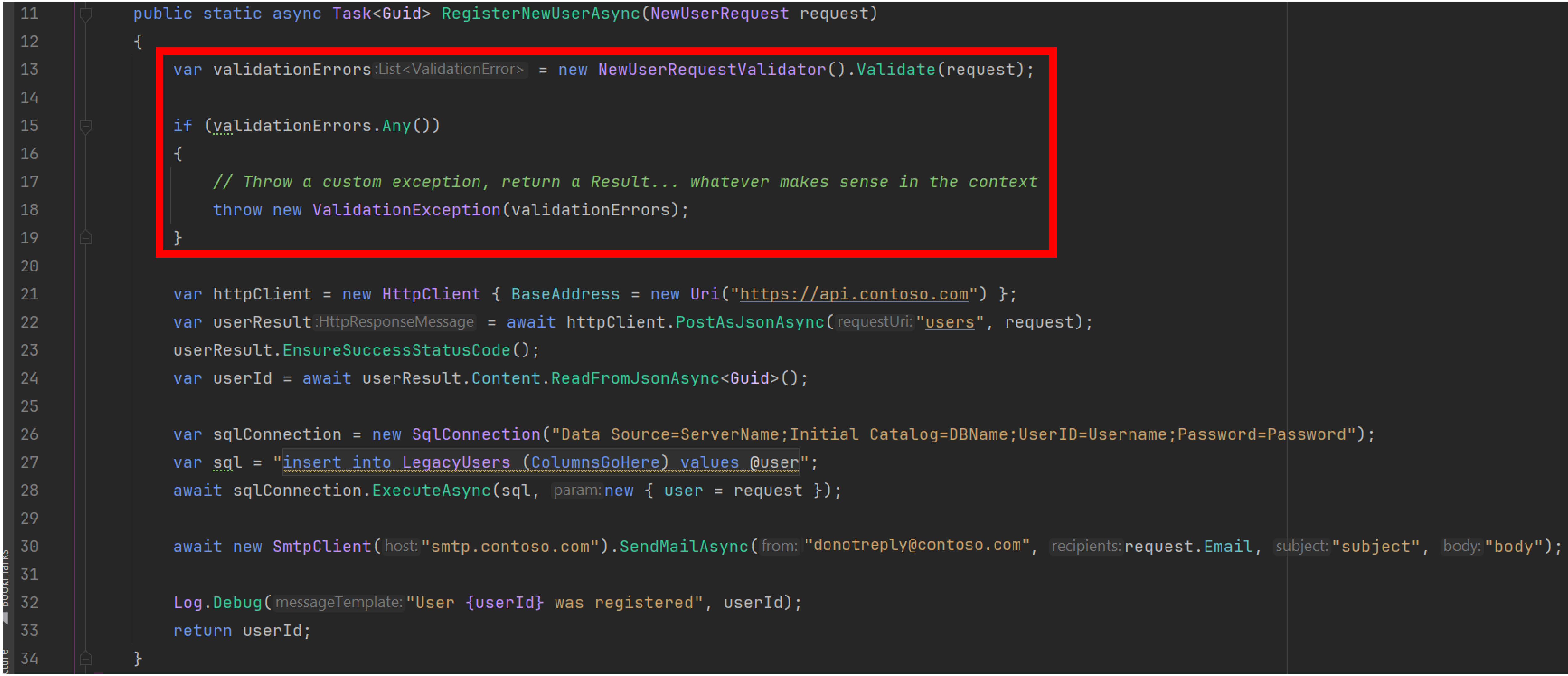
Sprout Example – New Code Tested In Isolation



```
public class NewUserRequestValidator
{
    1 usage
    public List<ValidationError> Validate(NewUserRequest request)
    {
        List<ValidationError> errors = [];

        if (!new EmailAddressAttribute().IsValid(request.Email))
            errors.Add(item: new ValidationError(nameof(request.Email),  Error: "Email address is in an invalid format"));

        return errors;
    }
}
```

*just use Fluent Validation

Sprout Example – Sprout into Existing Code



```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    var validationErrors:List<ValidationError> = new NewUserRequestValidator().Validate(request);

    if (validationErrors.Any())
    {
        // Throw a custom exception, return a Result... whatever makes sense in the context
        throw new ValidationException(validationErrors);
    }

    var httpClient = new HttpClient { BaseAddress = new Uri("https://api.contoso.com") };
    var userResult:HttpResponseMessage = await httpClient.PostAsJsonAsync(requestUri:"users", request);
    userResult.EnsureSuccessStatusCode();
    var userId = await userResult.Content.ReadFromJsonAsync<Guid>();

    var sqlConnection = new SqlConnection("Data Source=ServerName;Initial Catalog=DBName;UserID=Username;Password=Password");
    var sql = "insert into LegacyUsers (ColumnsGoHere) values @user";
    await sqlConnection.ExecuteAsync(sql, param:new { user = request });

    await new SmtpClient(host:"smtp.contoso.com").SendMailAsync(from:"donotreply@contoso.com", recipients:request.Email, subject:"subject", body:"body");

    Log.Debug(messageTemplate:"User {userId} was registered", userId);
    return userId;
}
```

Sprout Takeaways

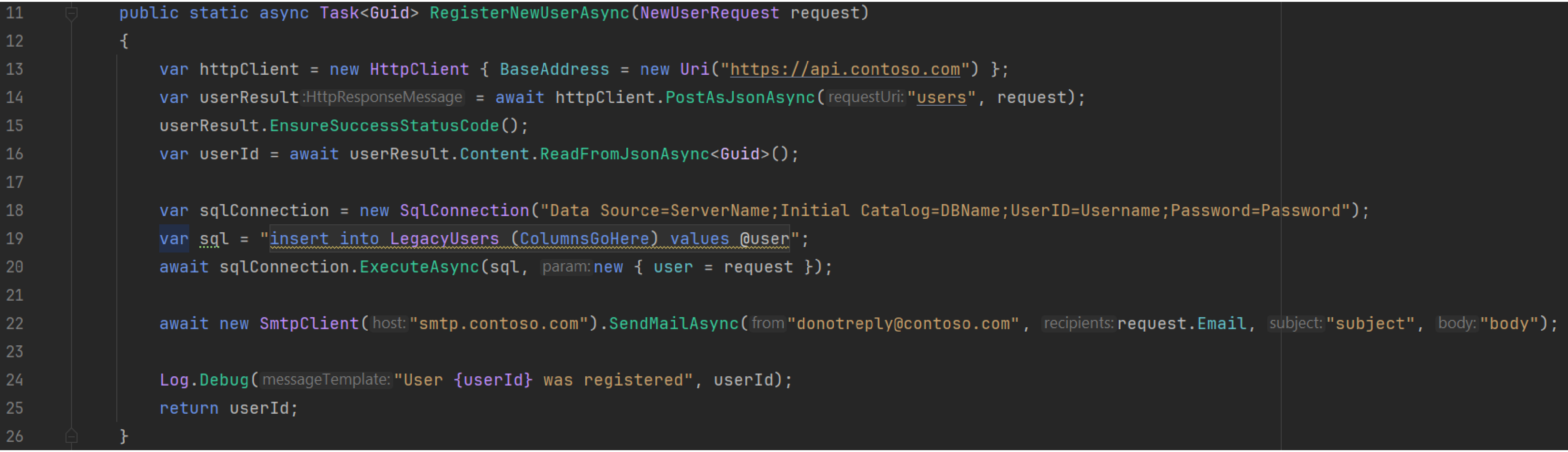
- Write new code in isolation from the mess
- If you keep adding to the pile of debt, the only way out is bankruptcy
- Eventually over time things will get better (might take months or years)
- Can even test most of the untested sprouted code in isolation!

Wrap

Wrap

- Only works if new code needs to be added to beginning or end of existing code
- Process is:
- Extract all existing code into another method (safe to do with an IDE)
- Call newly extracted method from the original method (also safe)
- Insert new code before/after the extracted method

Wrap Example – Existing Code



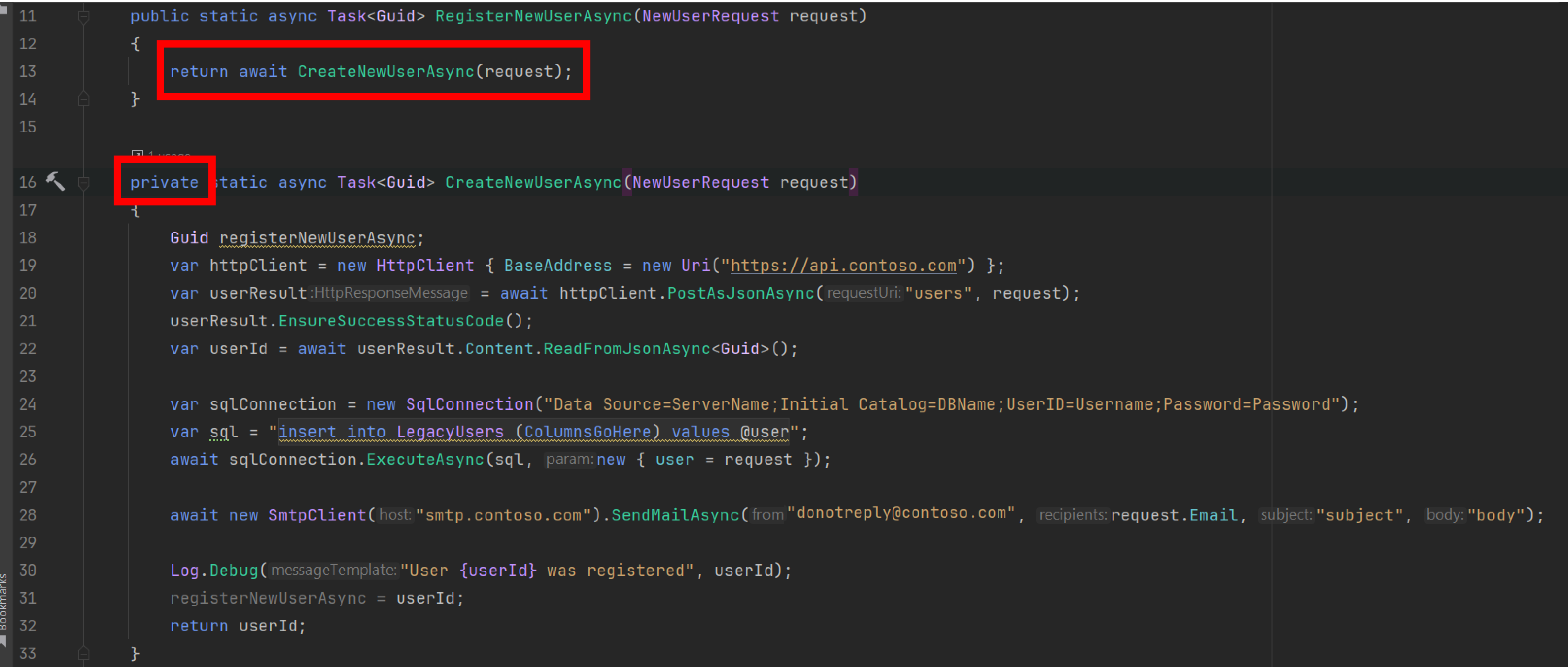
```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    var httpClient = new HttpClient { BaseAddress = new Uri("https://api.contoso.com") };
    var userResult:HttpResponseMessage = await httpClient.PostAsJsonAsync(requestUri:"users", request);
    userResult.EnsureSuccessStatusCode();
    var userId = await userResult.Content.ReadFromJsonAsync<Guid>();

    var sqlConnection = new SqlConnection("Data Source=ServerName;Initial Catalog=DBName;UserID=Username;Password=Password");
    var sql = "insert into LegacyUsers (ColumnsGoHere) values @user";
    await sqlConnection.ExecuteAsync(sql, param:new { user = request });

    await new SmtpClient(host:"smtp.contoso.com").SendMailAsync(from"donotreply@contoso.com", recipients:request.Email, subject:"subject", body:"body");

    Log.Debug(messageTemplate:"User {userId} was registered", userId);
    return userId;
}
```

scottsauber

New Requirement!

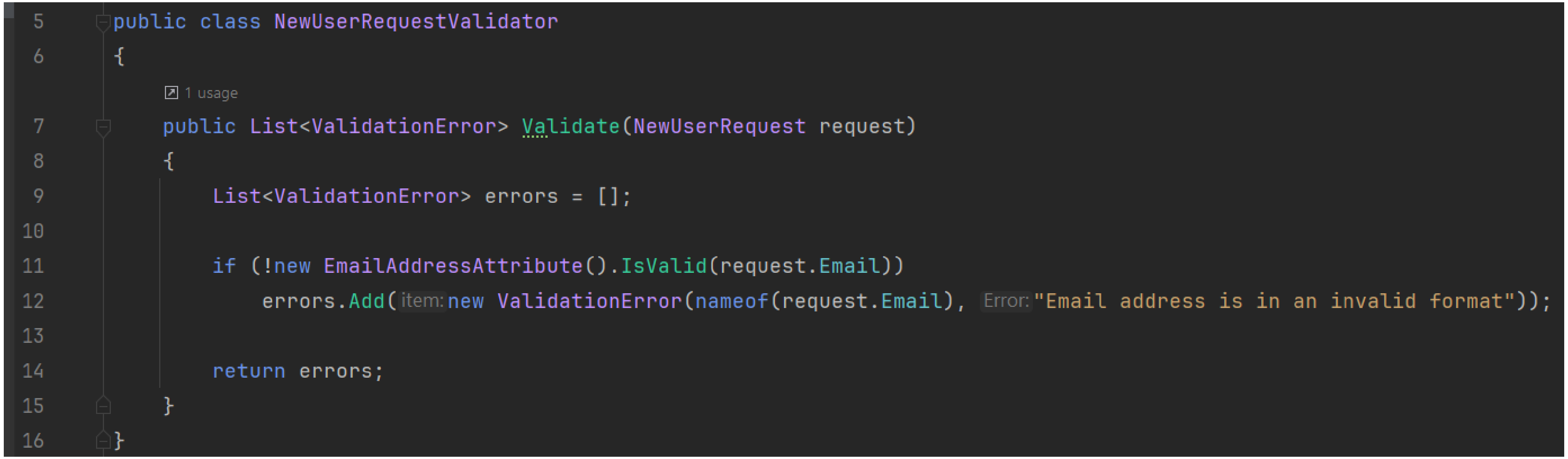
Do not allow an invalid email address to be registered

Wrap Step 1 –Extract To Private Method

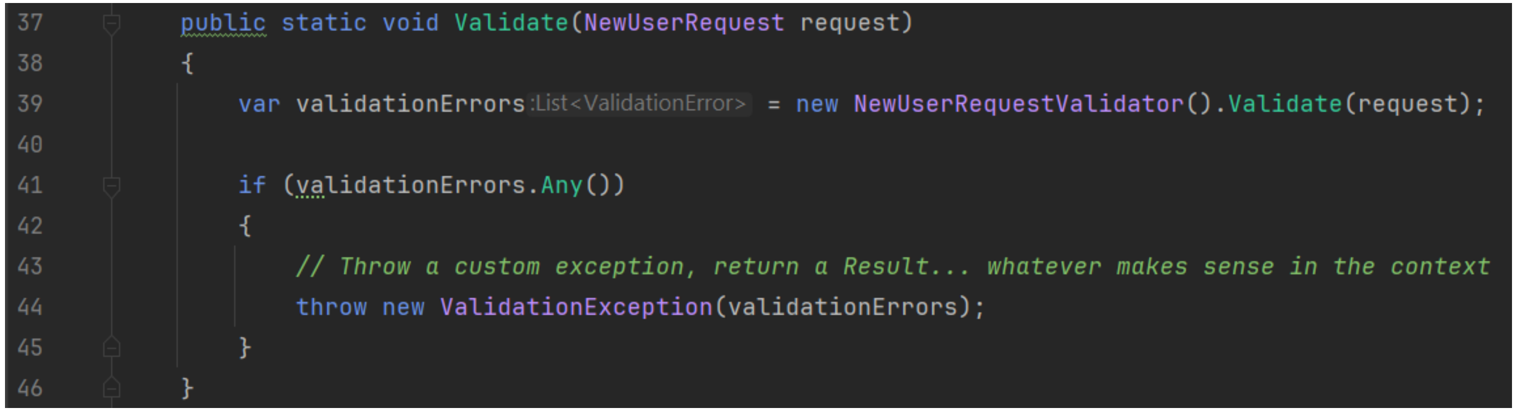


```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    return await CreateNewUserAsync(request);
}

1 usage
private static async Task<Guid> CreateNewUserAsync(NewUserRequest request)
{
    Guid registerNewUserAsync;
    var httpClient = new HttpClient { BaseAddress = new Uri("https://api.contoso.com") };
    var userResult:HttpResponseMessage = await httpClient.PostAsJsonAsync(requestUri: "users", request);
    userResult.EnsureSuccessStatusCode();
    var userId = await userResult.Content.ReadFromJsonAsync<Guid>();

    var sqlConnection = new SqlConnection("Data Source=ServerName;Initial Catalog=DBName;UserID=Username;Password=Password");
    var sql = "insert into LegacyUsers (ColumnsGoHere) values @user";
    await sqlConnection.ExecuteAsync(sql, param: new { user = request });

    await new SmtpClient(host: "smtp.contoso.com").SendMailAsync(from "donotreply@contoso.com", recipients: request.Email, subject: "subject", body: "body");

    Log.Debug(messageTemplate: "User {userId} was registered", userId);
    registerNewUserAsync = userId;
    return userId;
}
```

Wrap Step 2 – Create + Test New Methods



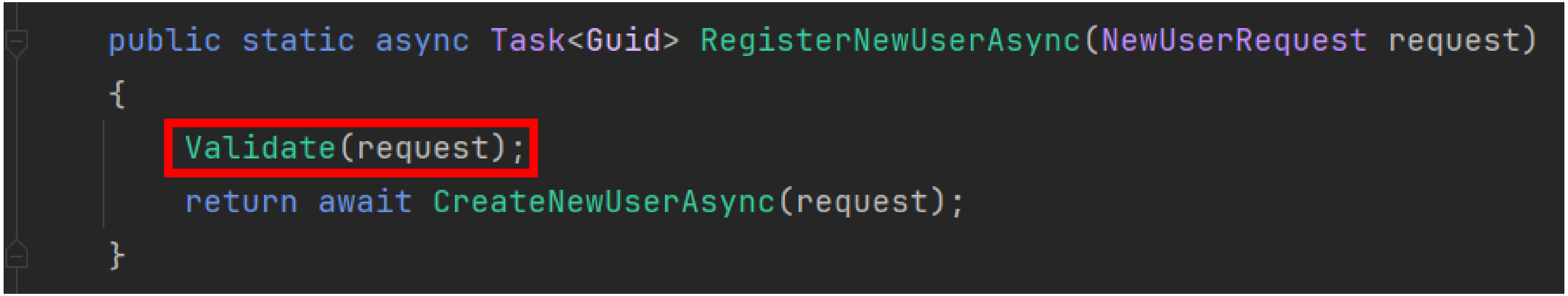
```
public class NewUserRequestValidator
{
    1 usage
    public List<ValidationError> Validate(NewUserRequest request)
    {
        List<ValidationError> errors = [];

        if (!new EmailAddressAttribute().IsValid(request.Email))
            errors.Add( item: new ValidationError(nameof(request.Email),  Error: "Email address is in an invalid format"));

        return errors;
    }
}
```



```
    public static void Validate(NewUserRequest request)
    {
        var validationErrors :List<ValidationError>  = new NewUserRequestValidator().Validate(request);

        if (validationErrors.Any())
        {
            // Throw a custom exception, return a Result... whatever makes sense in the context
            throw new ValidationException(validationErrors);
        }
    }
```

scottsauber

Wrap Step 3 – Call from Original Method



```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    Validate(request);
    return await CreateNewUserAsync(request);
}
```

scottsauber

Wrap Takeaways

- Similar to Sprout (especially this example)
- Only works in some scenarios (new code added to start/end)

Sprout and Wrap Takeaways

- Use these techniques when you need to make a change immediately
- Sprout works always
- Wrap can work when code can be added to beginning or end

Questions about Sprout or Wrap?

Question to ask yourself:
How much time do you have?

Answer 2:

We have some extra time!

Avoid The Great Rewrite™

- The Great Rewrite™ is $$ and risky ***especially without proper understanding of the current system***
- Delays new feature development
- No iterative steps
- Without tests, you are Editing and Praying
- Don’t try to rewrite the entire app
- Instead - slice off pieces

Avoid The Great Rewrite™

- …but before you carve off, understand current state
- Can’t properly rearchitect without understanding current slice
- I am usually not a fan of moving app logic + db schema at the same time
- Move the app logic first, then move the db schema later when everything is on the new code
- Avoid replicating data, whole new sets of problems

How do I evolve Legacy Code?

1. Get the developer ecosystem stable
2. Understand what the app does
3. Add characterization/integration tests around existing code
4. Refactor safely!

1. Get the Developer Ecosystem Stable

- Is it in source control?
- Is there a reliable CI pipeline?
- Is there a reliable CD pipeline?
- Can I run it locally without being pointed at Production?
- Can we remove our Production access?
- Is there reliable test data?

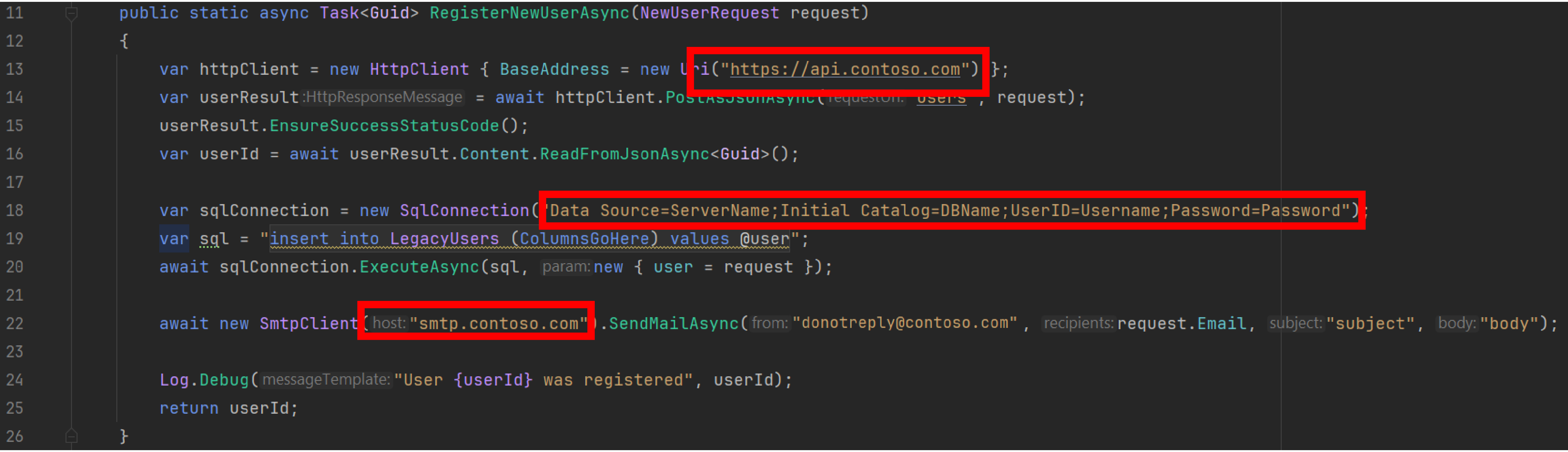
2. Understand the app

- Add logging
- Add metrics
- Add instrumentation
- Observe the app in Production
- Read the code and take notes
- Refactor without tests, but ***throw the code away when you’re done***

3. Add characterization tests

- After understanding the inputs and outputs, you can now add tests
- Blackbox implementation code as much as possible
- WebApplicationFactory is great for this in .NET
- Builds up knowledge of how the code works
- Need safety net when you refactor in the next step

3. Add characterization tests Paradox!



```
public static async Task<Guid> RegisterNewUserAsync(NewUserRequest request)
{
    var httpClient = new HttpClient { BaseAddress = new Uri("https://api.contoso.com") };
    var userResult :HttpResponseMessage = await httpClient.PostAsJsonAsync( requestUri: "users", request);
    userResult.EnsureSuccessStatusCode();
    var userId = await userResult.Content.ReadFromJsonAsync<Guid>();

    var sqlConnection = new SqlConnection("Data Source=ServerName;Initial Catalog=DBName;UserID=Username;Password=Password");
    var sql = "insert into LegacyUsers (ColumnsGoHere) values @user";
    await sqlConnection.ExecuteAsync(sql,  param: new { user = request });

    await new SmtpClient( host: "smtp.contoso.com").SendMailAsync( from: "donotreply@contoso.com",  recipients: request.Email,  subject: "subject",  body: "body");

    Log.Debug( messageTemplate: "User {userId} was registered", userId);
    return userId;
}
```

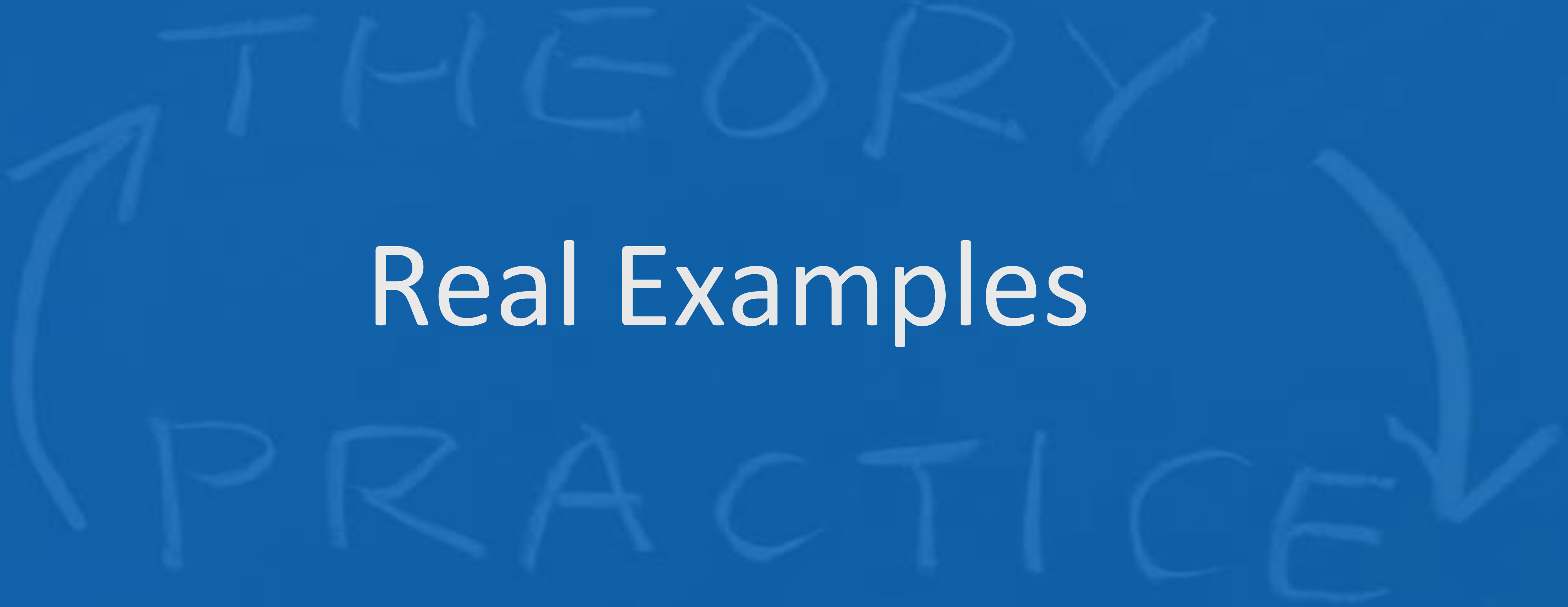
3. Add characterization tests Paradox

- Paradox!
- I need to add tests so I can change the code safely
- I can't test without changing the code
- Make ***absolute bare minimum changes*** to allow testability
- DI is your friend (constructor, method, and property injection!)

4. Refactor the Code

- Now that you have tests you can safely refactor
- Make change, run tests, repeat
- Get back to green as quickly as possible
- Avoid “long red”
- Repeat until code is in good state

Real Examples



Legacy Code Disclaimer

- Legacy code is not the fault of an individual
- Code is the product of an organization not an individual
  - Time constraints
  - Technical constraints
  - Technical training (or lack thereof)
  - Allowing silos to form

Situation 1

- Inherit a weekly legacy ETL process
- ETL process sends data to hundreds of retail stores
- When the stores don’t have this data, they can’t set accurate prices
- 80% of this code is SQL (including xp_cmdshell), 10% is VB, 10% is C#
- Takes 40 hours to run and fails 90% of the time, needs babysat
- Required 20+ hours of off-hours support from devs, every single week
- Key Dev making this work left the company
- Millions of dollars lost to the business not having accurate data
- Tiger Team Created to fix this

1. Get The Developer Ecosystem Stable

- Created CI/CD Pipelines
- Got it to run locally (without pointing at Production)
- Root caused issues and fixed
- Avoided extending the timeout on long running stored procedures
- On-call rotation of 2 to shield team from immediate production issues

2. Understand the application

- Refactored w/out committing
- Created tools/utilized Grafana to measure the ETL worked
- Add instrumentation to understand how it performed in Production (NewRelic)
- Zero domain knowledge from 90% of team members
- Pair Programmed

3. Add characterization tests

- Created test environment
- Dockerized the databases required
- Added integration tests
- First test required 40K LOC of SQL across 3 databases
- Created tool to autogenerate that 40K LOC so we didn’t have to maintain it

4. Refactor

- Analyzed NewRelic instrumentation
- Identified bottlenecks
- Optimizing code now safe to make with integration tests in place

Outcomes

- ETL process went from running 40hrs to 1hr
- 20 hours of off hours support dropped to 0
- Stores get their data faster
- Code is easier to make changes
- Users reported longstanding bugs, because they expected things to work

Situation 2

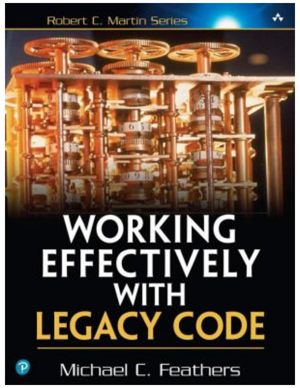
- Inherited WPF .NET 4.6 app that was not reliable
- Desire to move it to the web
- Added characterization tests
- Converted it to latest .NET version
- Moved code to shared library
- Created web app
- Moved page by page over
- Reduced dozens of known issues with manual intervention to zero

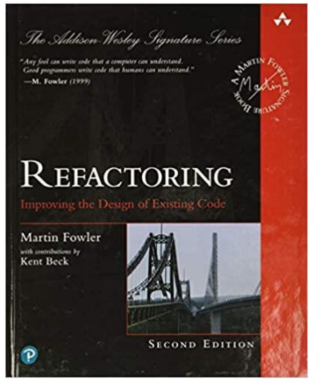
Takeaways

- Evolving should be your default, not rewriting
- Sprout or Wrap are techniques to deliver value quickly
- 4 step process when you do have time to “do it right”
- The payoff

Resources

- The Book
  - Working Effectively with Legacy Code by Michael Feathers
  - 435 pages
- https://understandlegacycode.com/
- Refactoring: Improving the Design of Existing Code by Martin Fowler
- This slide deck




The Addison-Wesley Signature Series
REFACTORING
Improving the Design of Existing Code
Martin Fowler
with contributions by Kent Beck
SECOND EDITION

Questions?

Contact: ssauber@leantechniques.com

Ask Me Anything

Slides at scottsauber.com

Thanks!

Slides up at scottsauber.com

scottsauber