Unit Testing im Brownfield?

Challenge Accepted!

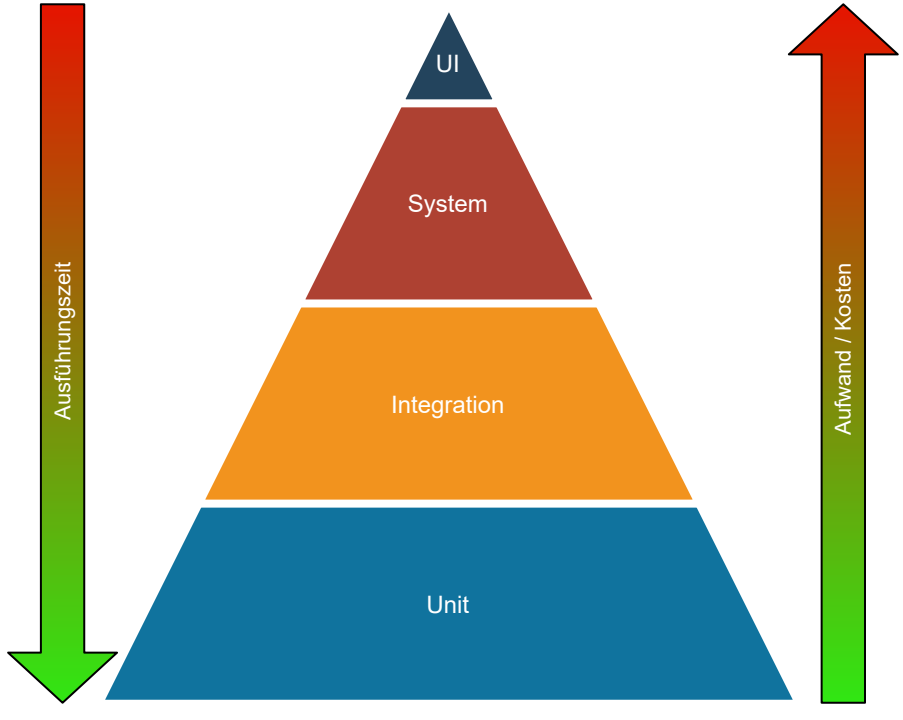
Andreas Richter

Stand der Dinge

Warum automatisiert testen?

- Funktionalität durch automatisierte Tests absichern
- Ständige Überprüfung gegenüber Spezifikationen
- Sicherheitsnetz aufbauen
    - für Erweiterungen
    - für Restrukturierungen

Was gibt es für Tests?



Warum sind gewachsene Systeme schwer testbar?

- Code über mehrere Jahre gewachsen
- Code über mehrere Entwicklergenerationen gewachsen
- Viel Code unter Zeit- und Budgetdruck entstanden
- Bekannte Patterns nicht eingehalten
- Resultat:
  - ~~Historisch~~ Hysterisch gewachsenes System
  - Unstrukturierter Code
  - Fragmentierte Domänenlogik im Code

Beispiel

```
private async void ButtonDelete_Click(object sender, RoutedEventArgs e)
{
    if (MessageBox.Show("Soll die Session wirklich gelöscht werden?",
        "Bestätigen", MessageBoxButton.YesNo) == MessageBoxResult.Yes)
    {
        if (int.TryParse(TextBoxId.Text, out var id))
        {
            var apiClient = new ApiClient();
            var deleted = await apiClient.DeleteSession(id);
            MessageBox.Show(deleted ?
                "Session erfolgreich gelöscht." :
                "Session konnte nicht gelöscht werden.", "Info");
            NewSession();
            await LoadSessions();
        }
    }
}
```

Code direkt in Button-Click-Methode (Code behind)
Direkte Abhängigkeiten (new ApiClient, Messagebox.Show)

Beispiel

```
1  private async void ButtonSave_Click(object sender, RoutedEventArgs e)
2  {
3      if (string.IsNullOrEmpty(TextBoxTitle.Text))
4      {
5          MessageBox.Show("Es muss ein Titel angegeben werden.");
6          return;
7      }
8
9      if (string.IsNullOrEmpty(TextBoxAbstract.Text))
10     {
11         MessageBox.Show("Es muss ein Abstract angegeben werden.");
12         return;
13     }
14
15     ...
16     var saved = await apiClient.Save(session);
17     ...
18 }
```

Validierungen im Code Behind

Beispiel

```
1  private async void ButtonSave_Click(object sender, RoutedEventArgs e)
2  {
3      ...
4      if (int.TryParse(TextBoxId.Text, out var id))
5      {
6          var session = new Session
7          {
8              Id = id,
9              Title = TextBoxTitle.Text,
10             Abstract = TextBoxAbstract.Text
11         };
12
13         var apiClient = new ApiClient();
14         var saved = await apiClient.Save(session);
15         MessageBox.Show(saved ?
16             "Session erfolgreich gespeichert." :
17             "Session konnte nicht gespeichert werden.", "Info");
18         await LoadSessions();
19     }
20 }
```

Beispiel

```
1 [HttpPost]
2 public async Task<ActionResult<Session>> PostSession(Session session)
3 {
4     var context = new ConferenceContext();
5     context.Sessions.Add(session);
6     await context.SaveChangesAsync();
7
8     return CreatedAtAction("GetSession", new { id = session.Id }, session);
9 }
```

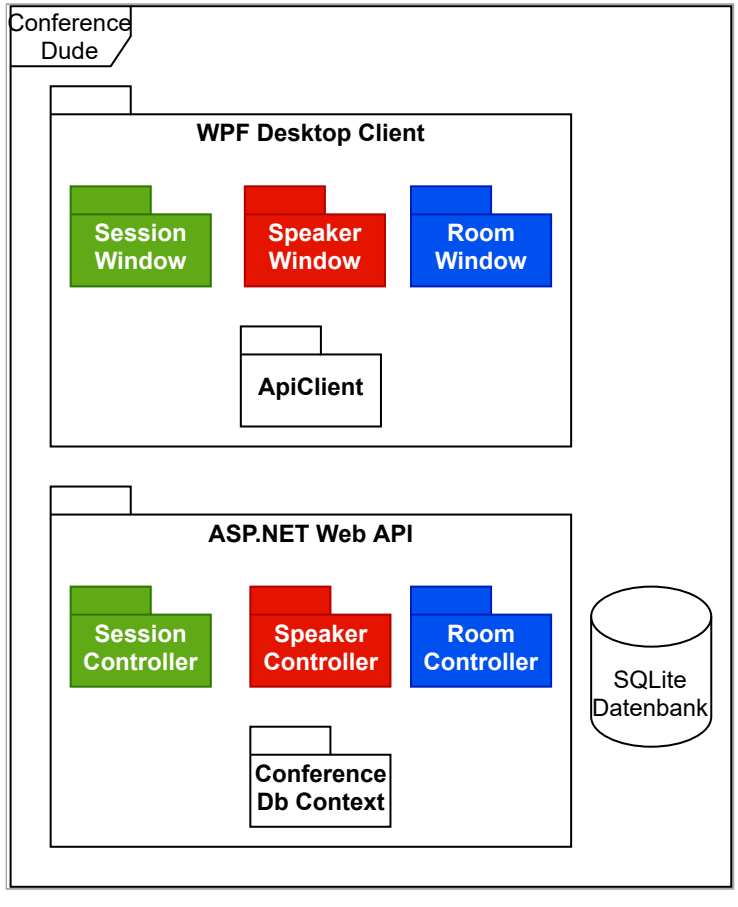
DB-Aufruf aus API Controller heraus

Was muss überhaupt getestet werden?

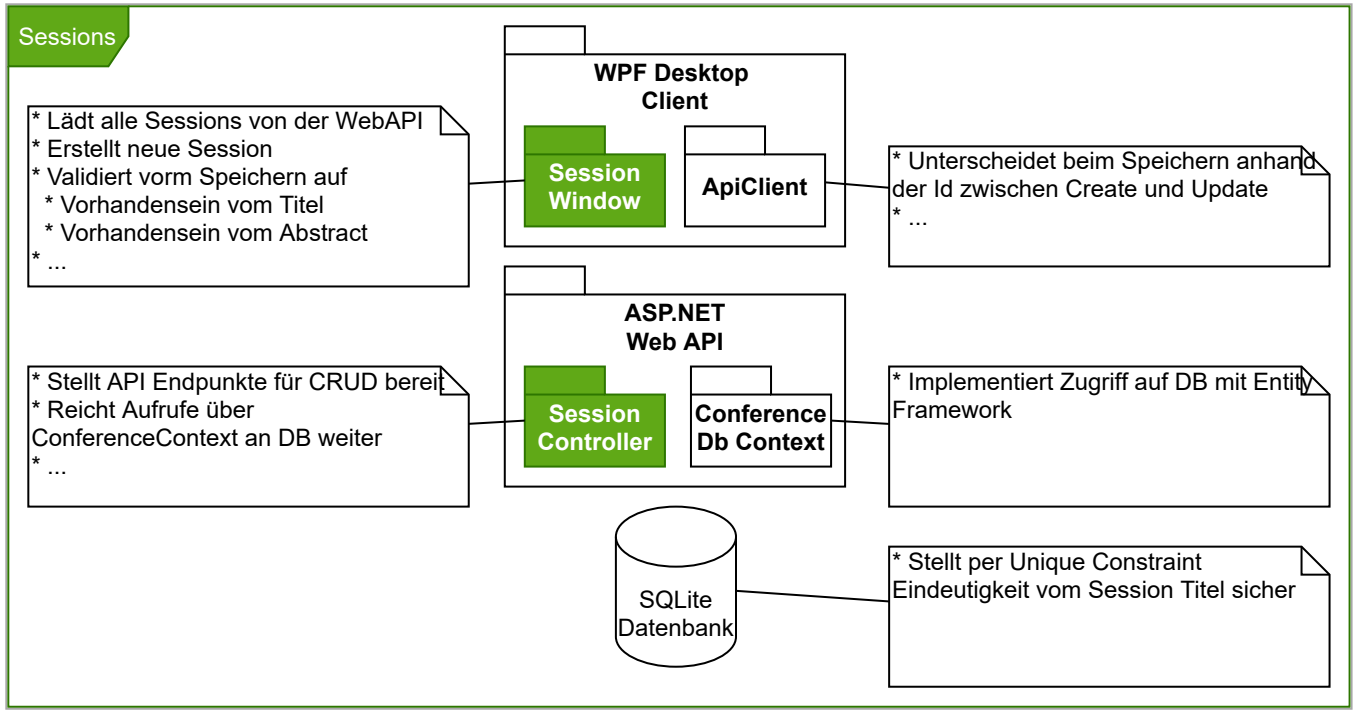
System analysieren

- Fachliche Module erkennen
- Codeverteilung pro Modul erforschen
- Domänenlogik pro Modul extrahieren
- Datenbank nicht vergessen!

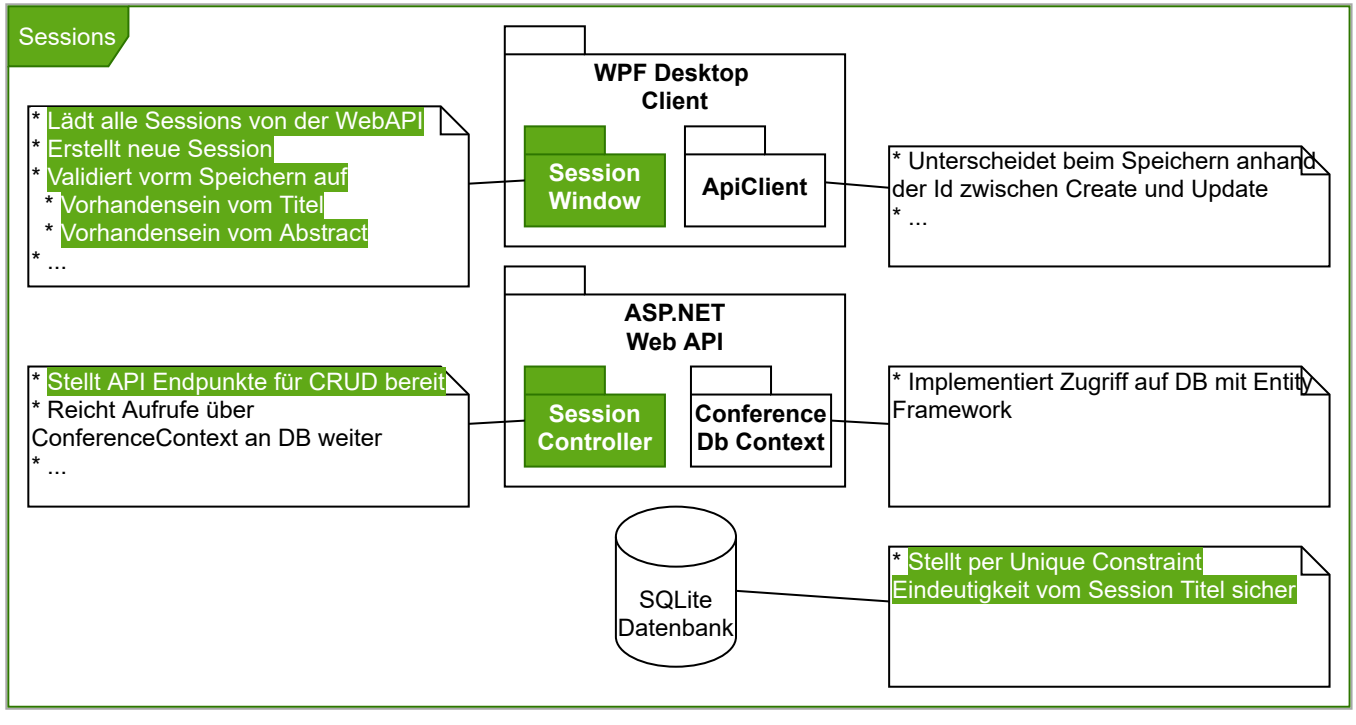
Fachliche Module erkennen



Codeverteilung pro Modul erforschen

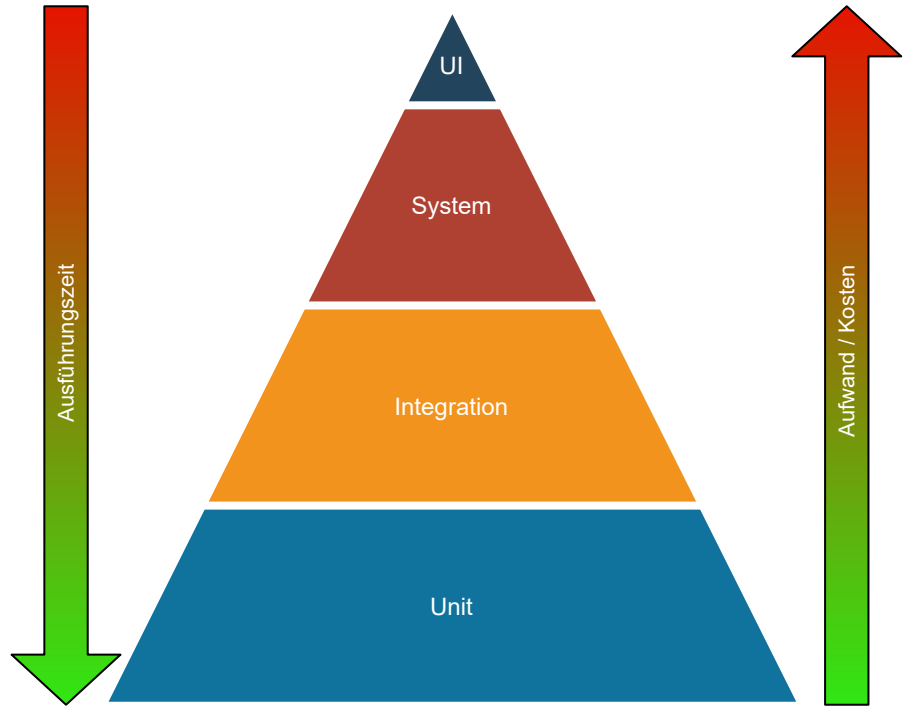


Domänenlogik pro Modul extrahieren



Was muss überhaupt getestet werden?

1. Domänenlogik - Unit Tests
2. Zusammenarbeit von Client & Server mit Domänenlogik - Integrationstests
3. Integration von der UI / API bis zur Datenbank - Systemtests
4. Bedienbarkeit vom UI - UI Tests (automatisiert / teilautomatisiert)
5. Optik - UI Tests (manuell / teilautomatisiert)



Wie wird die Domänenlogik getestet?

Sprout Method

- Greenfield Spross im Brownfield Sumpf
- Neue Funktionalität testgetrieben entwickeln
- Alter Code nutzt neuen Spross
- Nicht umgekehrt!

Sprout - Domain Model

```
1  public class Session
2  {
3      public int Id { get; set; }
4      public string Title { get; set; }
5      public string Abstract { get; set; }
6
7      public ValidationResult Validate()
8      {
9          var validationResult = new ValidationResult();
10
11         if (string.IsNullOrEmpty(Title))
12             validationResult.AddError(nameof(Title), "Das Feld ist ein Pflichtfeld")
13         if (string.IsNullOrEmpty(Abstract))
14             validationResult.AddError(nameof(Abstract), "Das Feld ist ein Pflichtfe
15
16         return validationResult;
17     }
18 }
```

Sprout - Verwendung im Client vorher

```
1  private async void ButtonSave_Click(object sender, RoutedEventArgs e)
2  {
3      if (string.IsNullOrEmpty(TextBoxTitle.Text))
4      {
5          MessageBox.Show("Es muss ein Titel angegeben werden.");
6          return;
7      }
8
9      if (string.IsNullOrEmpty(TextBoxAbstract.Text))
10     {
11         MessageBox.Show("Es muss ein Abstract angegeben werden.");
12         return;
13     }
14     ...
15 }
```

Sprout - Verwendung im Client nachher

```
1  private async void ButtonSave_Click(object sender, RoutedEventArgs e)
2  {
3      if (int.TryParse(TextBoxId.Text, out var id))
4      {
5          var sessionModel = new SessionModel
6          {
7              Id = id,
8              Title = TextBoxTitle.Text,
9              Abstract = TextBoxAbstract.Text
10         };
11
12         var validationResult = sessionModel.ToSession().Validate();
13         if (!validationResult.Success)
14         {
15             foreach (var message in validationResult.Messages)
16             {
17                 MessageBox.Show(
18                     $"Feld {message.FieldName} - {message.ErrorMessage}");
19             }
20
21             return;
```

Sprout - Domain Service

```
1  public class SessionService
2  {
3      ...
4      public async Task<(ValidationResult validationResult, int id)> Create(Session se
5      {
6          var validationResult = new ValidationResult();
7          var createdSessionId = 0;
8
9          var existingSession = await _sessionRepository.GetByTitle(session.Title);
10         if (existingSession != null)
11         {
12             validationResult.AddError(
13                 nameof(Session.Title),
14                 "Eine Session mit diesem Titel ist bereits vorhanden.");
15         }
16         else
17         {
18             createdSessionId = await _sessionRepository.Create(session).ConfigureAwa
19         }
20
21         return (validationResult, createdSessionId);
```

Sprout - Verwendung im Server vorher

```
1 [HttpPost]
2 public async Task<ActionResult<Session>> PostSession(Session session)
3 {
4     var context = new ConferenceContext();
5     context.Sessions.Add(session);
6     await context.SaveChangesAsync();
7 
8     return CreatedAtAction("GetSession", new { id = session.Id }, session);
9 }
```

Eindeutige Titel per Unique Constraint in der DB!

Sprout - Verwendung im Server nacher

```
1  [HttpPost]
2  public async Task<ActionResult<SessionDto>> PostSession(SessionDto session)
3  {
4      var createResult = await _sessionService.Create(session.ToSession()).ConfigureAw
5
6      if (createResult.validationResult.Success)
7      {
8          var newSession = await _sessionRepository.GetById(createResult.id).Configure
9          var newSessionDto = newSession.ToSessionDto();
10         return CreatedAtAction("GetSession", new { id = newSessionDto.Id }, newSessi
11     }
12
13     var modelStateDictionary = new ModelStateDictionary();
14     foreach (var message in createResult.validationResult.Messages)
15     {
16         modelStateDictionary.TryAddModelError(message.FieldName, message.ErrorMessag
17     }
18     return ValidationProblem(modelStateDictionary);
19 }
```

Sprout - Domain Repository Interface

```
1 public interface ISessionRepository
2 {
3     Task<IReadOnlyCollection<Session>> GetAll();
4     Task<Session> GetById(int id);
5     Task<Session> GetByTitle(string title);
6     Task<int> Create(Session session);
7 }
```

Sprout - Verwendung im Server vorher

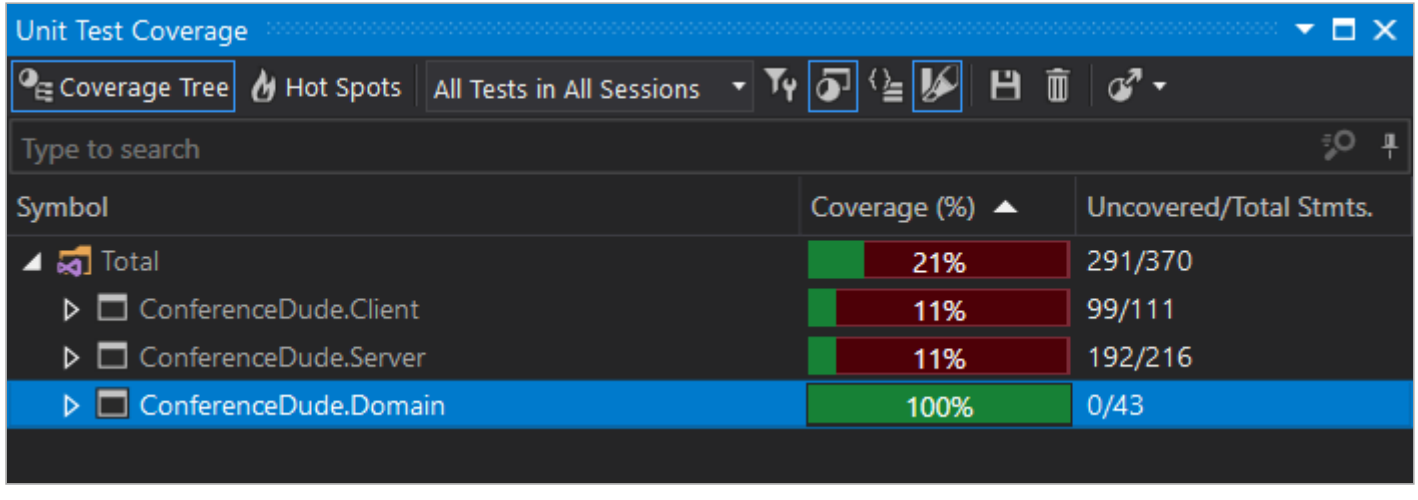
```
1 [HttpGet]
2 public async Task<ActionResult<IEnumerable<Session>>> GetSessions()
3 {
4     var context = new ConferenceContext();
5     return await context.Sessions.ToListAsync();
6 }
```

Sprout - Verwendung im Server nacher

```
1 [HttpGet]
2 public async Task<ActionResult<IEnumerable<SessionDto>>> GetSessions()
3 {
4     var sessions = await _sessionRepository.GetAll().ConfigureAwait(false);
5     var sessionDtos = sessions.Select(s => s.ToSessionDto()).ToList();
6     return sessionDtos;
7 }
```

Sprout - Code Coverage

- Domain 100% Abdeckung
- Client & Server weniger



Wie teste ich Client & Server?

Anwenden bekannter Patterns

- MVVM - Model View ViewModel
- MVP - Model View Presenter
- MVC - Model View Controller
- DI - Dependency Injection

SOLID Prinzipien anwenden

- Single Responsible Principle
  - Kleinere, leichter zu testende Klassen
- Interface Segregation Principle
  - Kleinere Interfaces, weniger Mockingaufwand
- Dependency Inversion Principle
  - Abhängigkeiten injizieren, Test in Isolation möglich

Recap

Vorgehen

1. Analysieren
2. Domänenlogik extrahieren
3. Sprout mit Domänenlogik testgetrieben entwickeln
4. Alten Code schrittweise umstellen
    - Verwenden vom Sprout
    - Patterns für bessere Testbarkeit
5. Goto 1.
    - Iterativ arbeiten

Starte mit fachlich einfachen Modulen!

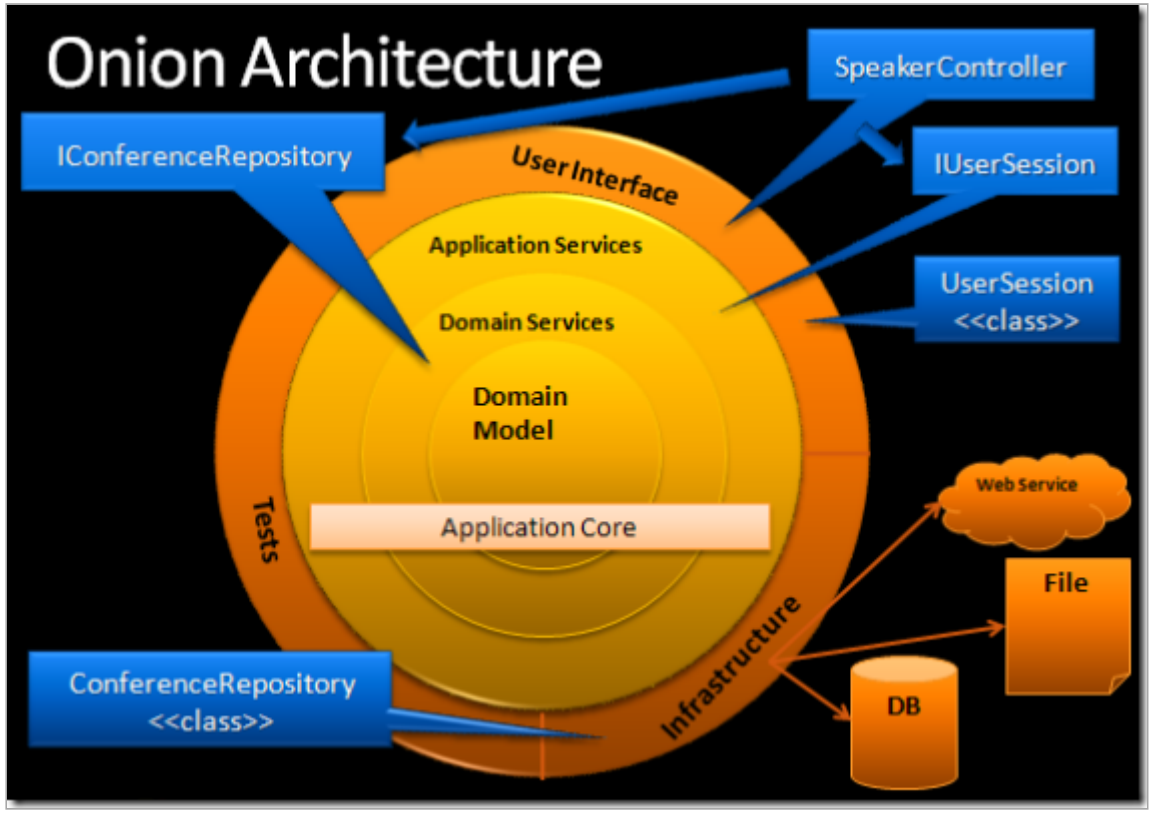
Domänenlogik mit DDD Bausteinen

- Domänenklassen
  - inklusive Logik auf dem Domänenobjekt selber
- Repository (Interface)
  - Definition der Persistenz von Domänenobjekten
- Services
  - Logik, die nicht in Domänenklassen passt

Pattern ”Ports & Adapters” verwenden

- Repository Interface in Domänenlogik
  - definiert Persistenz als Domänenlogik
- Implementierung für Produktiv
  - mit Zugriff auf DB
  - per Adapter im Server
- Implementierung für Testfall
  - als InMemory Repository
  - mithilfe von Substitute / Mock
  - per Adapter im Testcode

Zielarchitektur



Auch bekannt als:

- Clean Architecture
- Hexagonal Architecture

Gibt es da noch mehr?

- Analyse Patterns (arc42)
- Improve Patterns (arc42)
- Design Patterns - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Working Effectively with Legacy Code - Robert C. Feathers
- Domain-Driven Design kompakt - Vaughn Vernon, Carola Lilienthal
- Langlebige Software-Architekturen - Carola Lilienthal

Vielen Dank!

Andreas Richter

Software Craftsman & Architect

-  ar@anrichter.net
-  @anrichter
-  anrichter
-  https://anrichter.net