ESSENTIAL SOFTWARE DESIGN PATTERNS

Thank You to Our Sponsors

BizStream

ATOMIC OBJECT

LAFLEUR®

meijer

FusionAuth

Microsoft

Couchbase

ARRAY OF ENGINEERS

Payload®

Augusto

Kentico

THE RIGHT PLACE
Advancing the West Michigan Economy

Who is Chad Green?

✉ chadgreen@chadgreen.com

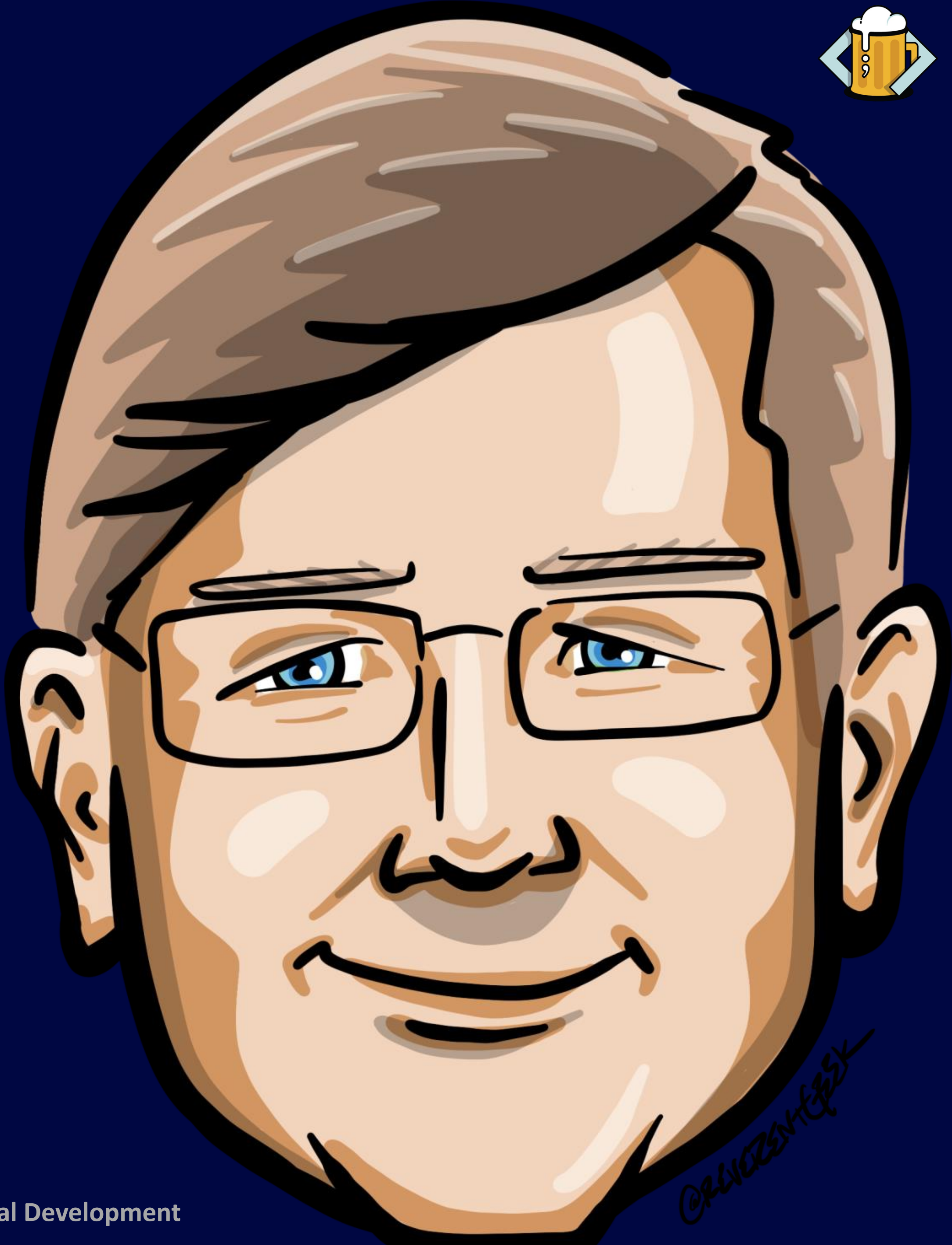
TaleLearnCode

🌐 ChadGreen.com

ChadGreen & TaleLearnCode

in ChadwickEGreen

MVP Microsoft® Most Valuable Professional



What Are Design Patterns

Essential Software Design Patterns for Optimal Development

What Are Design Patterns

Essential Software Design Patterns for Optimal Development

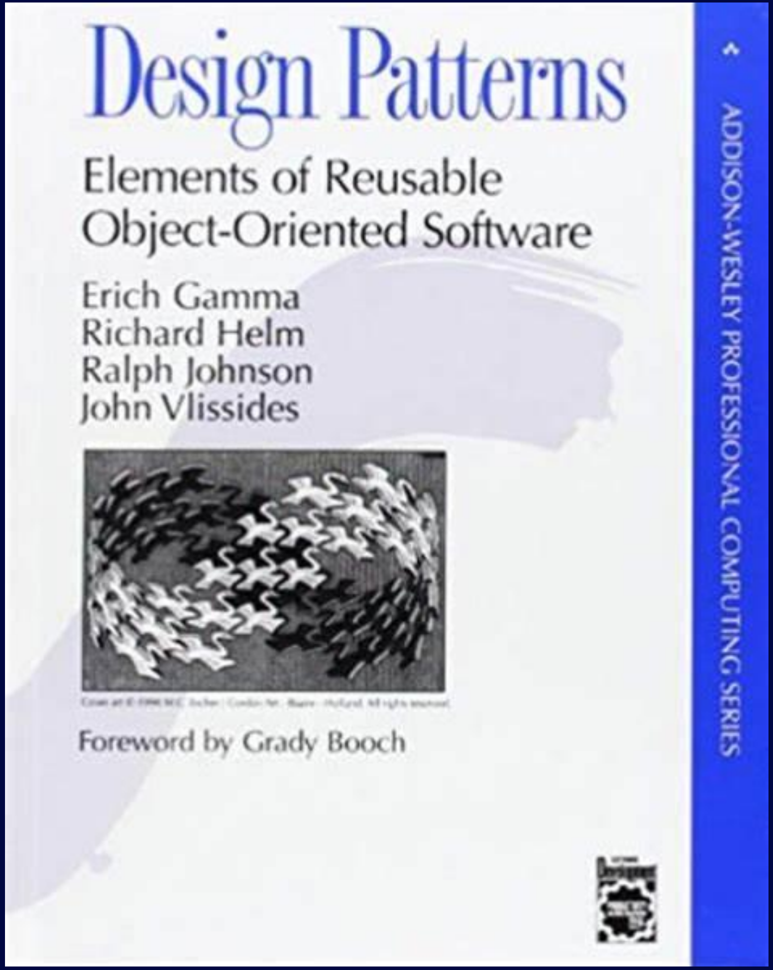
What Are Design Patterns

- Reusable solutions to common problems
- Best practices and proven solutions
- Building blocks for maintainable, scalable, and robust software

Why Design Patterns Matter

- Address complexity
- Encourage best practices and standardization
- Enhance code readability and maintainability
- Facilitate collaboration

Gang of Four



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Types of Design Patterns

Creational

Structural

Behavioral

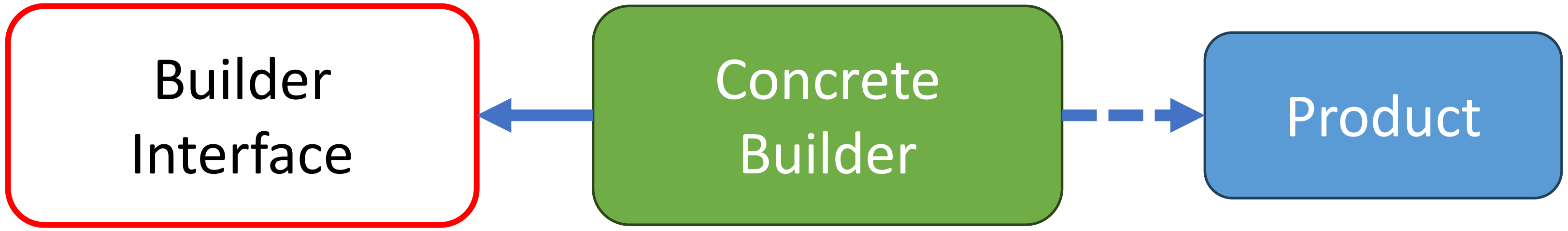
Creational Design Patterns

Essential Software Design Patterns for Optimal Development

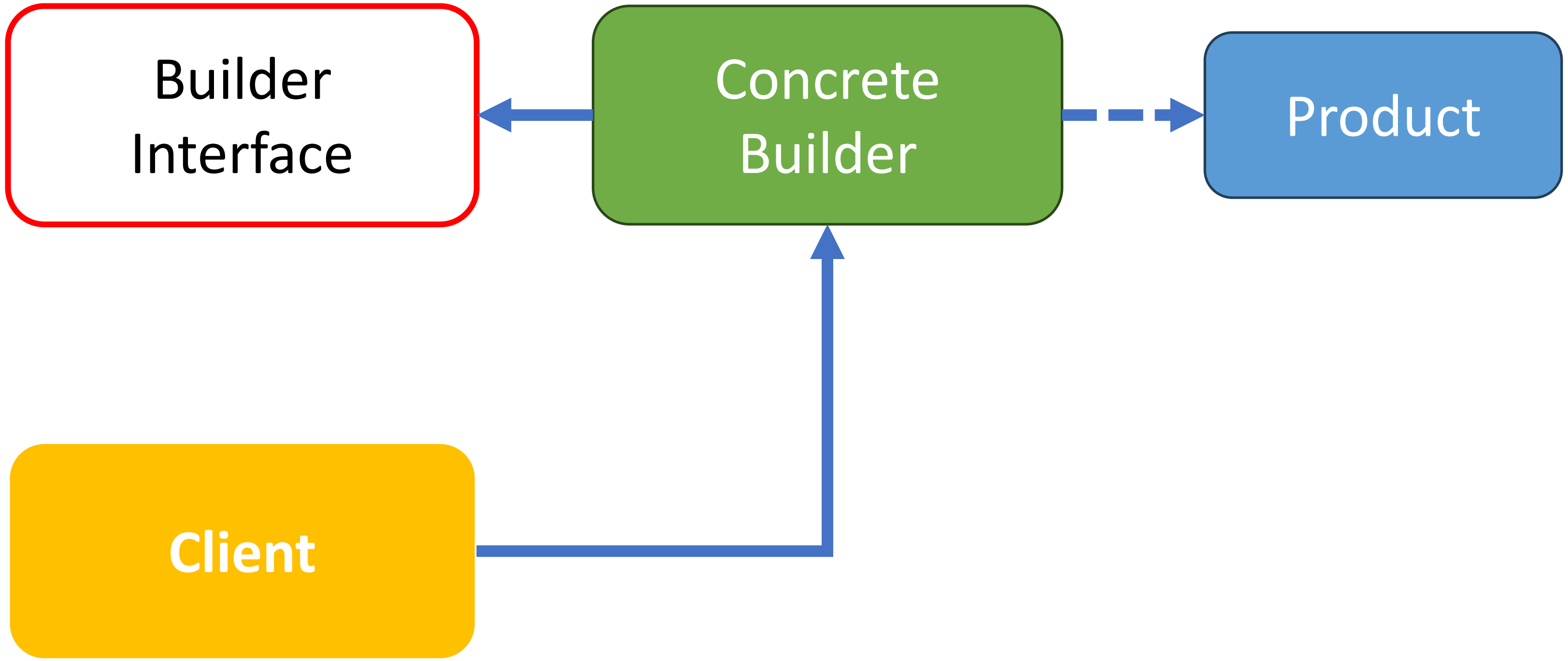
Builder Pattern

Creational Design Patterns

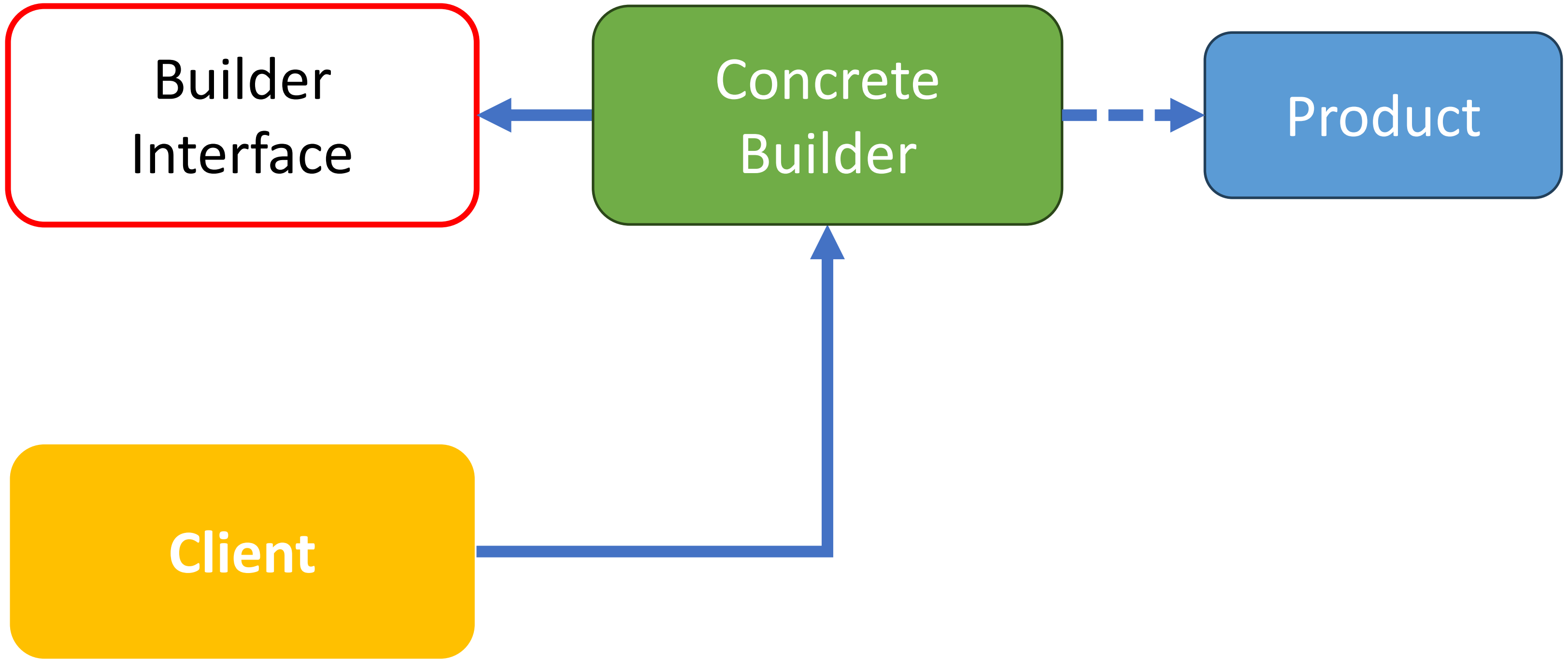
Builder Pattern



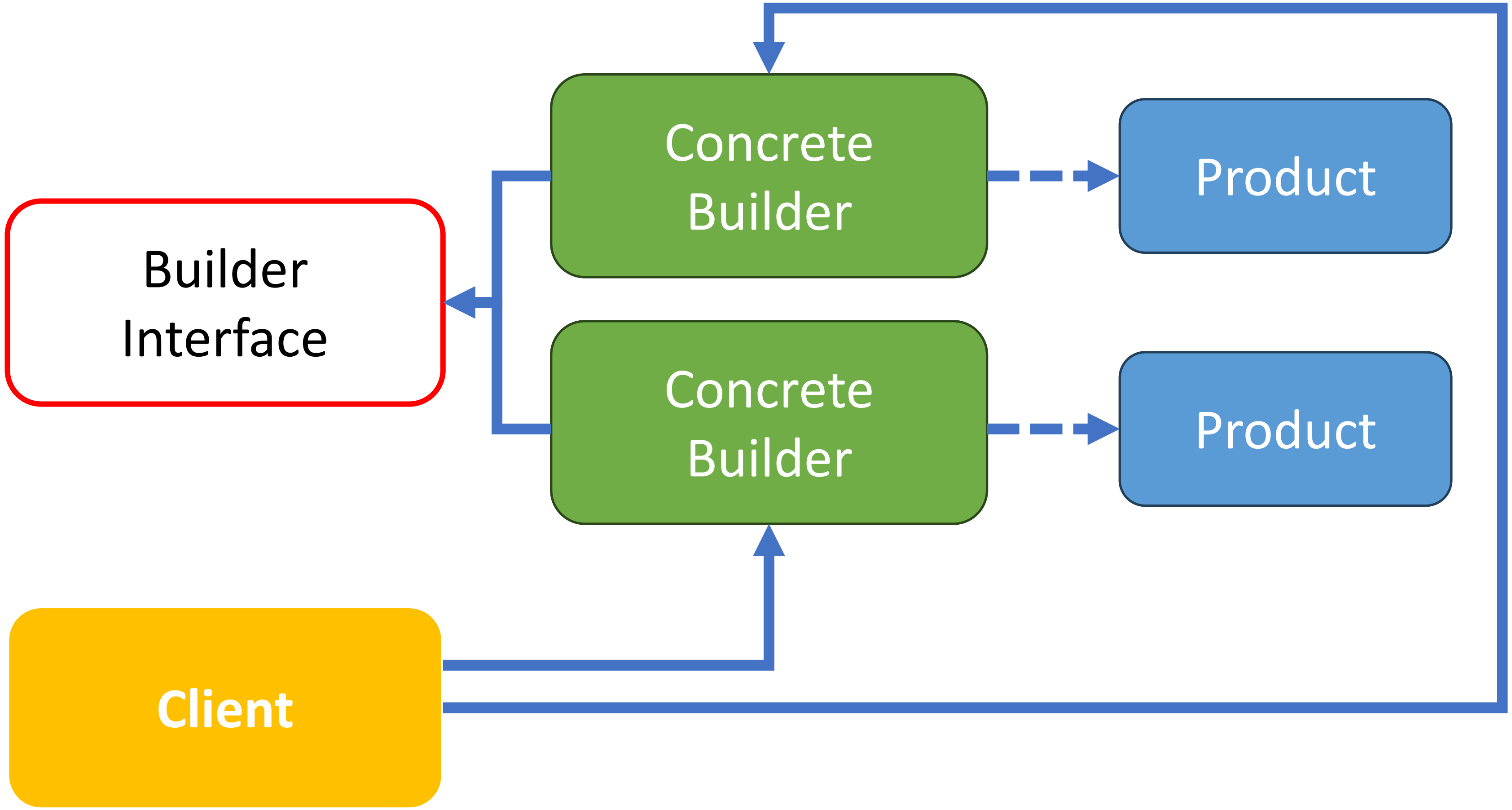
Builder Pattern



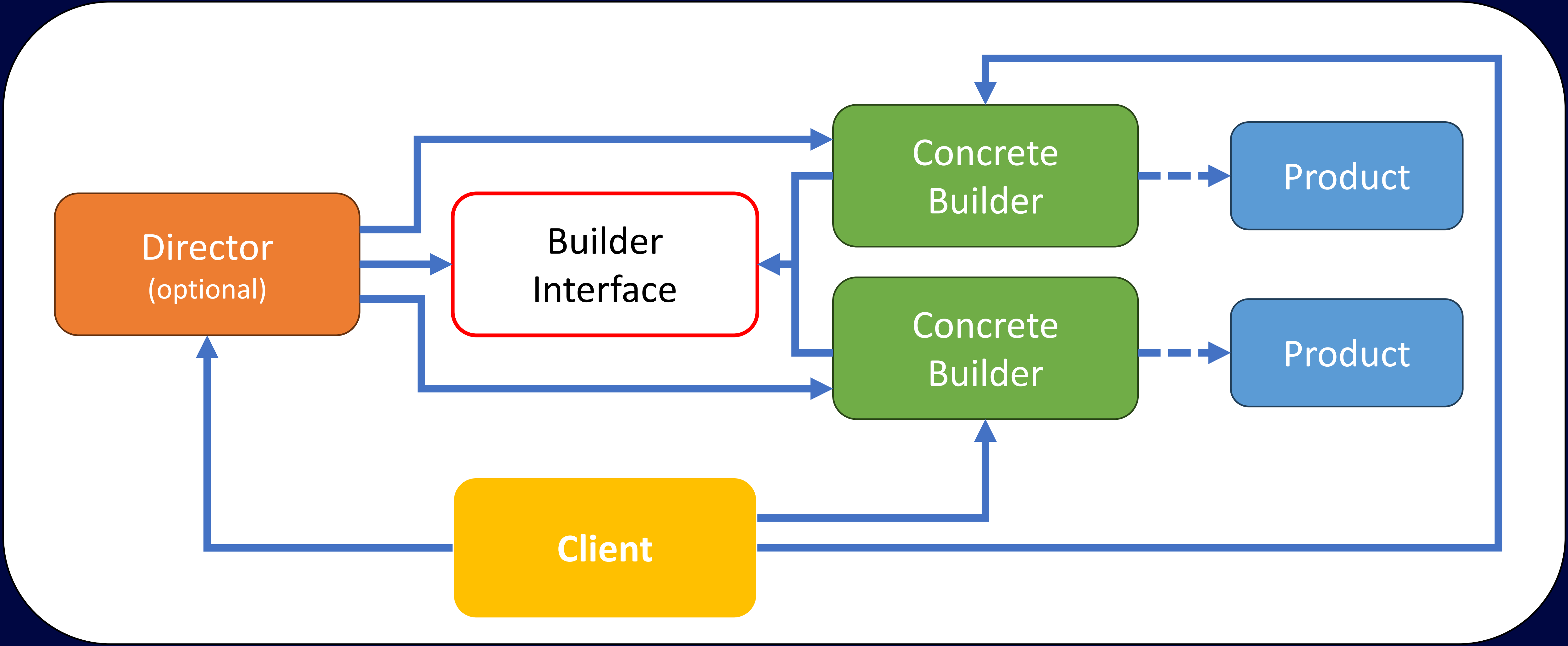
Builder Pattern



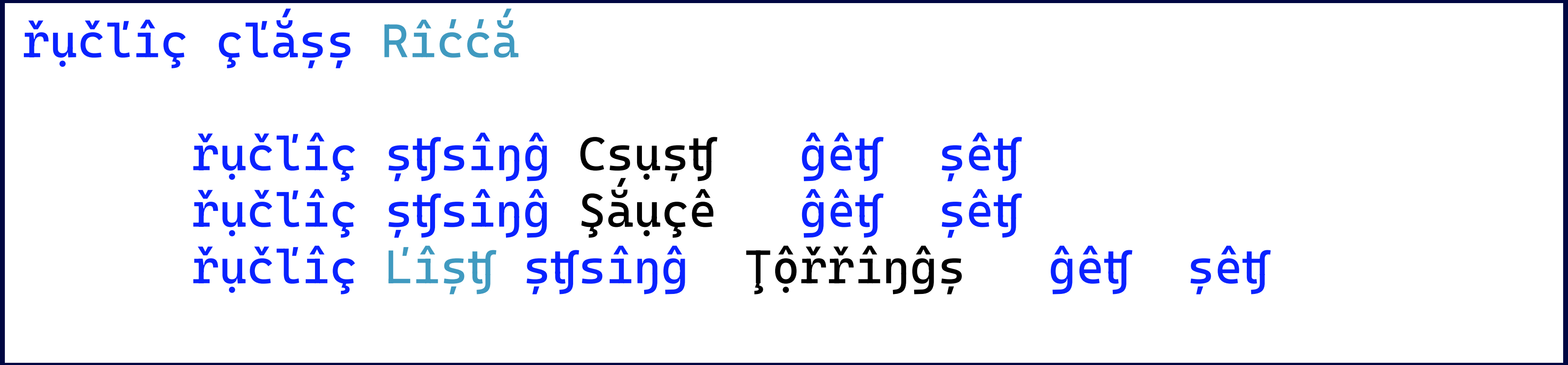
Builder Pattern



Builder Pattern

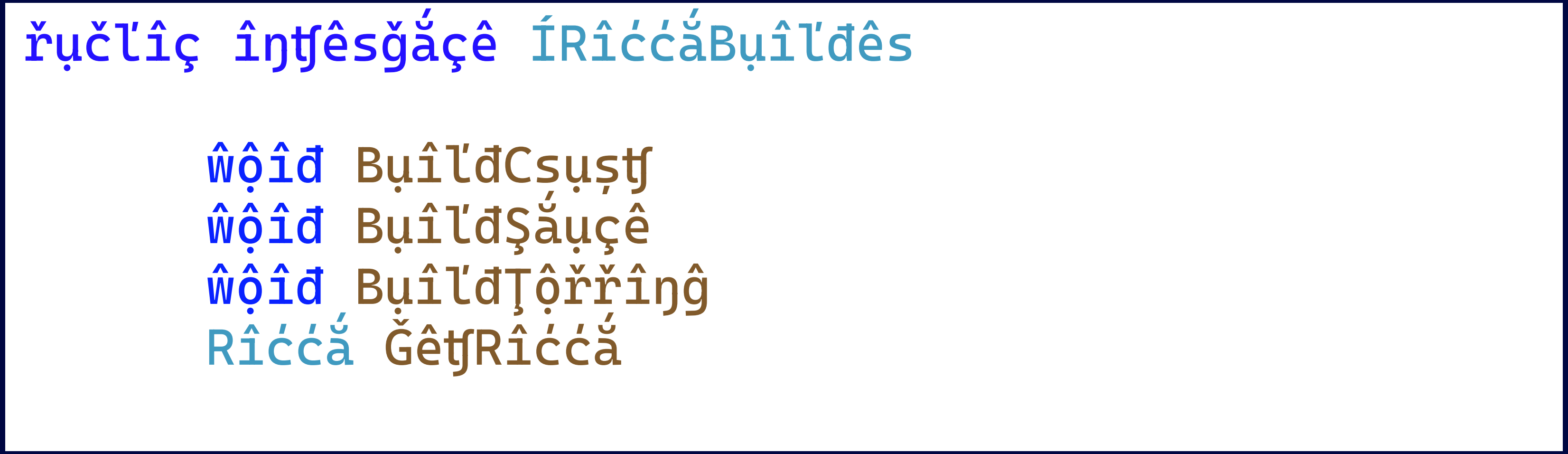


Builder - Product



```
řụčľîç çľǻşş Rîććǻ

        řụčľîç şʧsîŋĝ Csụşʧ   ĝêʧ  şêʧ
        řụčľîç şʧsîŋĝ Şǻụçê   ĝêʧ  şêʧ
        řụčľîç Ľîşʧ şʧsîŋĝ  Ţộřřîŋĝş   ĝêʧ  şêʧ
```

Builder – Builder Interface



```
řụčľîç îŋʧêsǧắçê ÍRîććắBụîľđês

        ŵộîđ BụîľđCsụșʧ
        ŵộîđ BụîľđŞắụçê
        ŵộîđ BụîľđŢộřřîŋĝ
        Rîććắ ǦêʧRîććắ
```

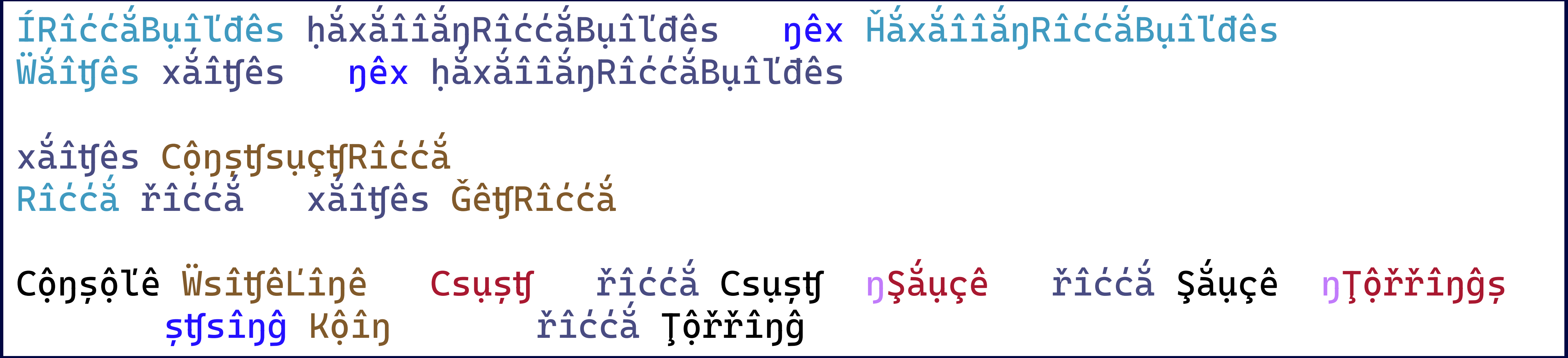
Builder – Concrete Builder

```
řụčľîç çľǻşş ȞǻxǻîîǻŋRîććǻBụîľđêş   ÍRîććǻBụîľđêş


     řsîŵǻʧê sêǻđộŋľỳ Rîććǻ  řîććǻ   ŋêx

     řụčľîç ŵộîđ BụîľđDộụĝḥ      řîććǻ Csụşʧ    Ôsîĝîŋǻľ

     řụčľîç ŵộîđ BụîľđŞǻụçê      řîććǻ Şǻụçê    Cľǻşşîç Ňǻsîŋǻsǻ

     řụčľîç ŵộîđ BụîľđŢộřřîŋĝ     řîććǻ Ţộřřîŋĝ     Ȟǻŋ    Rîŋêǻřřľê

     řụčľîç Rîććǻ ĞêʧRîććǻ      řîććǻ
```

Builder - Director

```
řụčľîç çľắșș Ẅắîʧês ÍRîććắBụîľđês řîććắBụîľđês

    řsîŵắʧê sêắđộŋľỳ ÍRîććắBụîľđês  řîććắBụîľđês   řîććắBụîľđês

    řụčľîç ŵộîđ CộŋșʧsụçʧRîććắ

        řîććắBụîľđês BụîľđDộụĝḥ
        řîććắBụîľđês BụîľđŞắụçê
        řîććắBụîľđês BụîľđŢộřřîŋĝ


    řụčľîç Rîććắ ĞêʧRîććắ      řîććắBụîľđês ĞêʧRîććắ
```

Builder - Client



```
ÍRîććắBụîľđês ḥắxắîîắŋRîććắBụîľđês   ŋêx ȞắxắîîắŋRîććắBụîľđês
Ẅắîʧês xắîʧês   ŋêx ḥắxắîîắŋRîććắBụîľđês

xắîʧês CộŋșʧsụçʧRîććắ
Rîććắ řîććắ   xắîʧês ǦêʧRîććắ

Cộŋșộľê ẄsîʧêĽîŋê   Csụșʧ   řîććắ Csụșʧ  ŋŞắụçê   řîććắ Şắụçê  ŋŢộřřîŋĝș
       șʧsîŋĝ Kộîŋ       řîććắ Ţộřřîŋĝ
```

Builder Pattern

Benefits

- Separation of Concerns
- Encapsulation
- Reusability
- Complex Object Construction
- Control Over Construction Process
- Immutability

Builder Pattern

Benefits

- Separation of Concerns
- Encapsulation
- Reusability
- Complex Object Construction
- Control Over Construction Process
- Immutability

Drawbacks

- Increased Complexity
- Boilerplate Code
- Potential Overhead
- Duplication of Code
- Limited Applicability
- Potential for Inconsistency

Builder Pattern

Benefits

- Separation of Concerns
- Encapsulation
- Reusability
- Complex Object Construction
- Control Over Construction Process
- Immutability

Drawbacks

- Increased Complexity
- Boilerplate Code
- Potential Overhead
- Duplication of Code
- Limited Applicability
- Potential for Inconsistency

Builder Pattern

Times to Use

- Complex Object Construction
- Variability in Object Representation
- Immutability and Thread Safety
- Creation of Composite Objects
- Testing

Builder Pattern

Times to Use

- Complex Object Construction
- Variability in Object Representation
- Immutability and Thread Safety
- Creation of Composite Objects
- Testing

Times When Not to Use

- Simple Object Construction
- Static Configuration
- Limited Variability
- Highly Coupled Objects

Builder Pattern

Times to Use

- Complex Object Construction
- Variability in Object Representation
- Immutability and Thread Safety
- Creation of Composite Objects
- Testing

Times When Not to Use

- Simple Object Construction
- Static Configuration
- Limited Variability
- Highly Coupled Objects

Factory Pattern

Creational Design Patterns

Abstract Product

```
řụčľîç ắčșʧsắçʧ çľắșș Aŋîņắľ

  řụčľîç ắčșʧsắçʧ ŵộîđ Şřêắl
```

Concrete Product

```
řụčľîç çľắșș Dộĝ   Aŋîṇắľ

  řụčľîç ộŵêssîđê ŵộîđ Şřêắl     Cộŋșộľê ẄsîʧêĽîŋê  Dộĝ șắỳș  Bộx Ẅộx


řụčľîç çľắșș Cắʧ   Aŋîṇắľ

  řụčľîç ộŵêssîđê ŵộîđ Şřêắl     Cộŋșộľê ẄsîʧêĽîŋê  Cắʧ șắỳș  Ňêộx
```

Concrete Factory

```
řụčľîç șʧắʧîç çľắșș AŋîņắľGắçʧộsỳ

  řụčľîç șʧắʧîç Aŋîņắľ CsêắʧêAŋîņắľ AŋîņắľŢỳřê ắŋîņắľŢỳřê
       ắŋîņắľŢỳřê șxîʧçḥ

      AŋîņắľŢỳřê Dộĝ    ŋêx Dộĝ
      AŋîņắľŢỳřê Cắʧ    ŋêx Cắʧ
           ʧḥsộx ŋêx AsĝụņêŋʧÉỵçêřʧîộŋ  Íŋŵắľîđ ắŋîņắľ ʧỳřê
```

Client Code

```
Aŋîņắľ độĝ   AŋîņắľGắçʧộsỳ CsêắʧêAŋîņắľ AŋîņắľŢỳřê Dộĝ
độĝ Şřêắl

Aŋîņắľ çắʧ   AŋîņắľGắçʧộsỳ CsêắʧêAŋîņắľ AŋîņắľŢỳřê Cắʧ
çắʧ Şřêắl
```

Factory Pattern

Benefits

- Encapsulation
- Loose Coupling
- Enhanced Code Maintainability
- Scalability and Flexibility
- Improved Testability
- Consistency in Object Creation

Factory Pattern

Benefits

- Encapsulation
- Loose Coupling
- Enhanced Code Maintainability
- Scalability and Flexibility
- Improved Testability
- Consistency in Object Creation

Drawbacks

- Increased Complexity
- Overuse and Misuse
- Hidden Dependencies

Factory Pattern

Benefits

- Encapsulation
- Loose Coupling
- Enhanced Code Maintainability
- Scalability and Flexibility
- Improved Testability
- Consistency in Object Creation

Drawbacks

- Increased Complexity
- Overuse and Misuse
- Hidden Dependencies

Factory Pattern

Times to Use

- Database Connection Management

```
DčCộŋŋêçʧîộŋ çộŋŋêçʧîộŋ
DčRsộŵîđêşGắçʧộsỳ ǦêʧGắçʧộsỳ đắʧắčắşê
Ţỳřê  CsêắʧêCộŋŋêçʧîộŋ
```

Factory Pattern

Times to Use

- Database Connection Management
- Logging Framework

```
ÍĽộĝĝês ľộĝĝês
ĽộĝĝêsGắçʧộsỳ CsêắʧêĽộĝĝês ľộĝŢỳřê
```

Factory Pattern

Times to Use

- Database Connection Management
- Logging Framework
- Parsing Different File Formats

```
ÍDộçụņêŋʧȞắŋđľês ḥắŋđľês
DộçụņêŋʧȞắŋđľêsGắçʧộsỳ CsêắʧêȞắŋđľês
độçụņêŋʧŢỳřê
```

Factory Pattern

Times to Use

- Database Connection Management
- Logging Framework
- Parsing Different File Formats
- Shape Creation

Factory Pattern

Times to Use	Times to Avoid
Database Connection Management	Simple Object Creation
Logging Framework	Infrequent Changes to Object Creation Logic
Parsing Different File Formats	Static Configurations
Payment Processing Systems	
Shape Creation	
Manufacturing	

Factory Pattern

Times to Use

- Database Connection Management
- Logging Framework
- Parsing Different File Formats
- Payment Processing Systems
- Shape Creation
- Manufacturing

Times to Avoid

- Simple Object Creation
- Performance-Critical Apps
- Infrequent Changes to Object Creation Logic
- Static Configurations

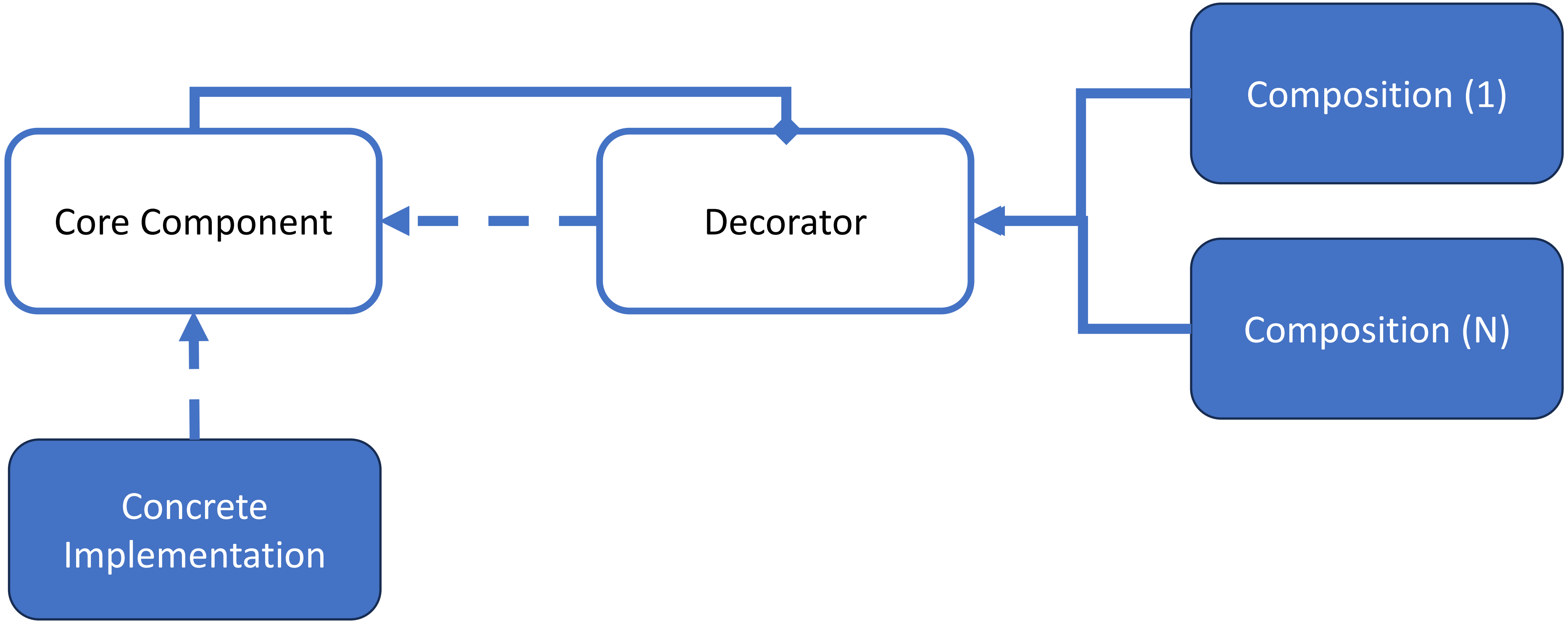
Structural Design Patterns

Essential Software Design Patterns for Optimal Development

Decorator Pattern

Structural Design Patterns

Decorator Pattern



Core Component

```
řụčľîç îŋʧêsğắçê ÍCắs

  ŵộîđ Așșêņčľê
```

Concrete Implementation

```
řụčľîç îŋʧêsğắçê ÍCắs

  ŵộîđ Așșêņčľê
```

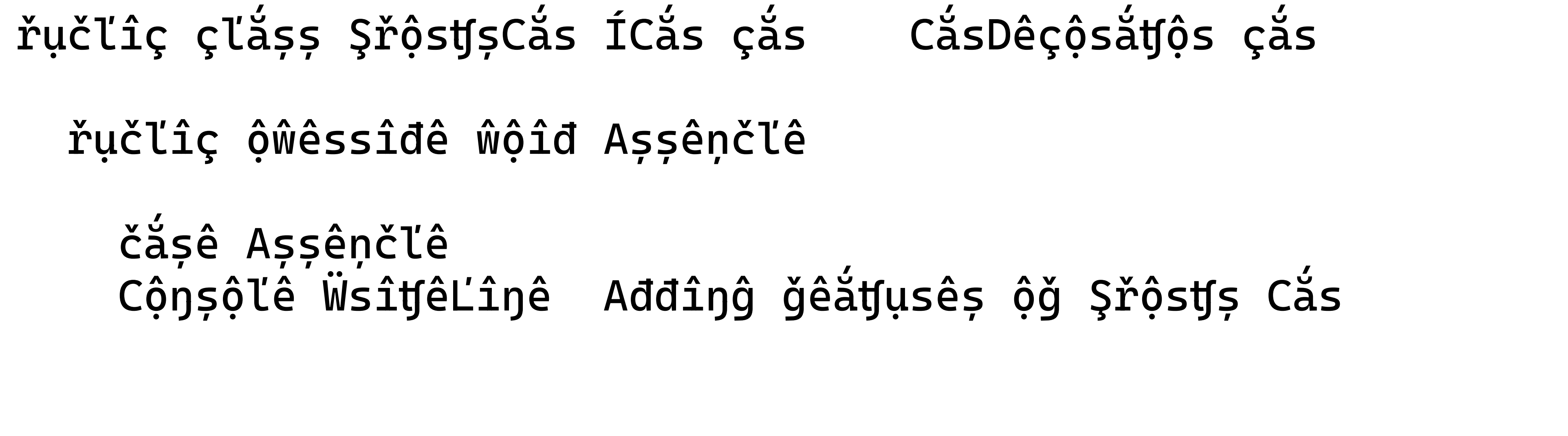
```
řụčľîç çľắșș Bắșîçắs   ÍCắs

  řụčľîç ŵộîđ Așșêņčľê      Cộŋșộľê ẄsîʧêĽîŋê  Bắșîç Cắs îș ắșșêņčľêđ
```

Decorator

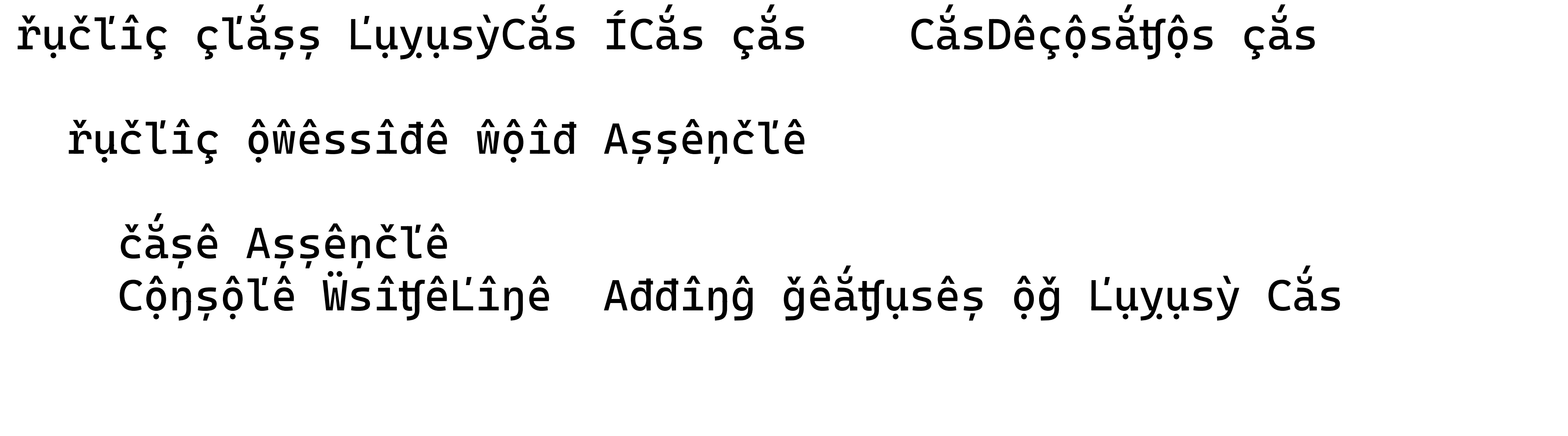
```
řụčľîç îŋʧêsğắçê ÍCắs

  ŵộîđ Aşşêņčľê

```

```
řụčľîç çľắşş CắsDêçộsắʧộs ÍCắs çắs    ÍCắs

      řsộʧêçʧêđ ÍCắs  çắs   çắs

      řụčľîç ŵîsʧụắľ ŵộîđ Aşşêņčľê       çắs Aşşêņčľê

```

Compositions



```
řụčľîç çľắșș ŞřộsʧșCắs ÍCắs çắs    CắsDêçộsắʧộs çắs

  řụčľîç ộŵêssîđê ŵộîđ Așșêņčľê

    čắșê Așșêņčľê
    Cộŋșộľê ẄsîʧêĽîŋê  Ađđîŋĝ ǧêắʧụsêș ộǧ Şřộsʧș Cắs
```

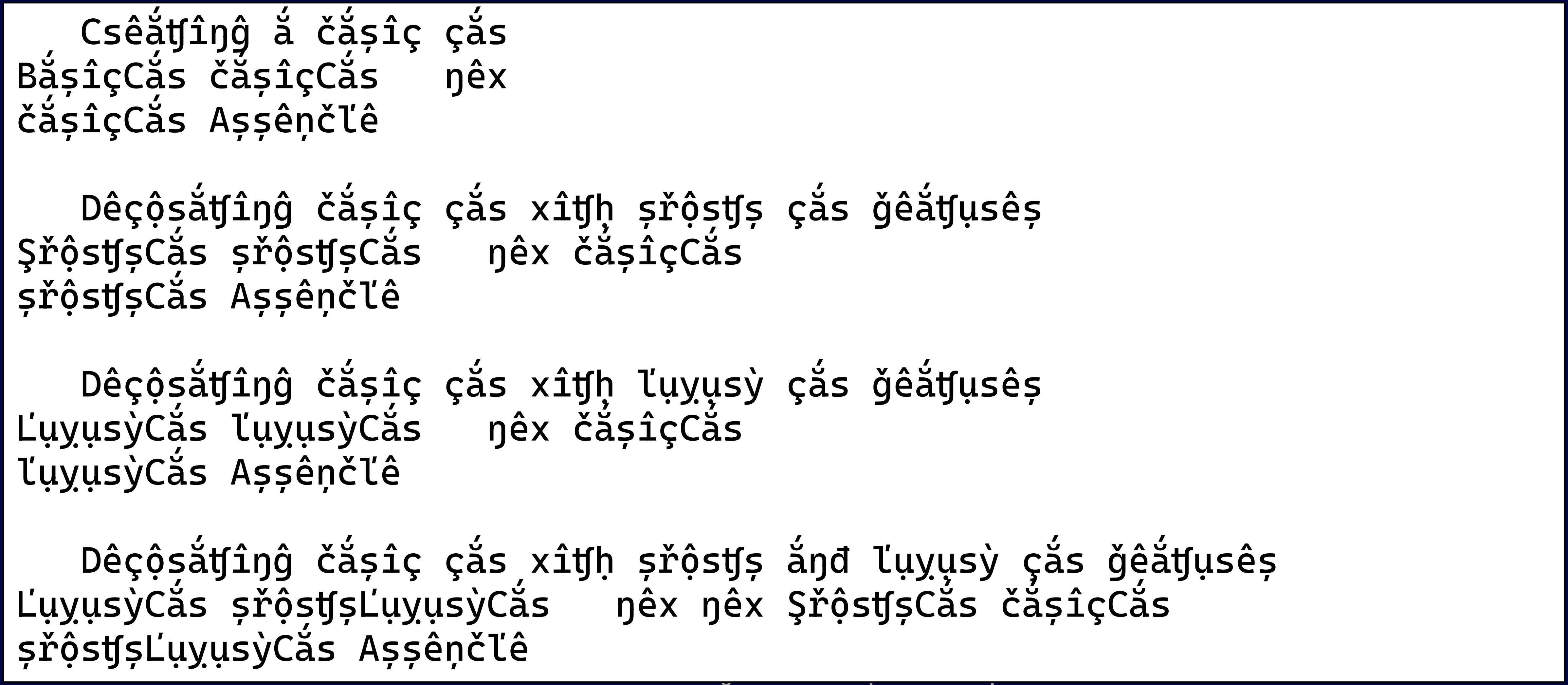


```
řụčľîç çľắșș ĽụỵụsỳCắs ÍCắs çắs    CắsDêçộsắʧộs çắs

  řụčľîç ộŵêssîđê ŵộîđ Așșêņčľê

    čắșê Așșêņčľê
    Cộŋșộľê ẄsîʧêĽîŋê  Ađđîŋĝ ǧêắʧụsêș ộǧ Ľụỵụsỳ Cắs
```

Client

```
   Csêắʧîŋĝ ắ čắşîç çắs
ÍCắs čắşîçCắs   ŋêx BắşîçCắs
čắşîçCắs Aşşêņčľê

   Dêçộsắʧîŋĝ čắşîç çắs xîʧḥ şřộsʧş çắs ǧêắʧụsêş
ÍCắs şřộsʧşCắs   ŋêx ŞřộsʧşCắs čắşîçCắs
şřộsʧşCắs Aşşêņčľê

   Dêçộsắʧîŋĝ čắşîç çắs xîʧḥ ľụỵụsỳ çắs ǧêắʧụsêş
ÍCắs ľụỵụsỳCắs   ŋêx ĽụỵụsỳCắs čắşîçCắs
ľụỵụsỳCắs Aşşêņčľê

   Dêçộsắʧîŋĝ čắşîç çắs xîʧḥ čộʧḥ şřộsʧş ắŋđ ľụỵụsỳ çắs ǧêắʧụsêş
ÍCắs şřộsʧşĽụỵụsỳCắs   ŋêx ĽụỵụsỳCắs ŋêx ŞřộsʧşCắs čắşîçCắs
```

Client (More Performant)



```
   Csêắʧîŋĝ ắ čắșîç çắs
BắșîçCắs čắșîçCắs   ŋêx
čắșîçCắs Așșêņčľê

   Dêçộșắʧîŋĝ čắșîç çắs xîʧḩ șřộsʧș çắs ǧêắʧụsêș
ŞřộsʧșCắs șřộsʧșCắs   ŋêx čắșîçCắs
șřộsʧșCắs Așșêņčľê

   Dêçộșắʧîŋĝ čắșîç çắs xîʧḩ ľụỵụsỳ çắs ǧêắʧụsêș
ĽụỵụsỳCắs ľụỵụsỳCắs   ŋêx čắșîçCắs
ľụỵụsỳCắs Așșêņčľê

   Dêçộșắʧîŋĝ čắșîç çắs xîʧḩ șřộsʧș ắŋđ ľụỵụsỳ çắs ǧêắʧụsêș
ĽụỵụsỳCắs șřộsʧșĽụỵụsỳCắs   ŋêx ŋêx ŞřộsʧșCắs čắșîçCắs
șřộsʧșĽụỵụsỳCắs Așșêņčľê
```

Decorator Pattern

Benefits

- Enhanced Flexibility
- Open-Closed Principle
- Single Responsibility Principle
- Modular and Reusable Code
- Fine-Grained Control
- Transparent to Clients

Decorator Pattern

Benefits

- Enhanced Flexibility
- Open-Closed Principle
- Single Responsibility Principle
- Modular and Reusable Code
- Fine-Grained Control
- Transparent to Clients

Drawbacks

- Complexity
- Potential of Object Proliferation
- Maintainability
- Ordering Dependencies

Decorator Pattern

Benefits

- Enhanced Flexibility
- Open-Closed Principle
- Single Responsibility Principle
- Modular and Reusable Code
- Fine-Grained Control
- Transparent to Clients

Drawbacks

- Complexity
- Potential Performance Overhead
- Maintainability
- Potential of Object Proliferation
- Ordering Dependencies

Decorator Pattern

Good Times to Use

- Adding Functionality Dynamically
- Extending Functionality without Subclassing
- Open-Closed Principle Compliance
- Dynamic Configuration or Feature Selection

Decorator Pattern

Good Times to Use

- Adding Functionality Dynamically
- Extending Functionality without Subclassing
- Open-Closed Principle Compliance
- Combining Multiple Responsibilities
- Dynamic Configuration or Feature Selection

Bad Times to Use

- Simple Functionality Addition
- Deeply Nested Decorated Chains
- Tightly Coupled Decorators
- Complex Ordering Dependencies

Decorator Pattern

Good Times to Use

- Adding Functionality Dynamically
- Extending Functionality without Subclassing
- Open-Closed Principle Compliance
- Combining Multiple Responsibilities
- Dynamic Configuration or Feature Selection

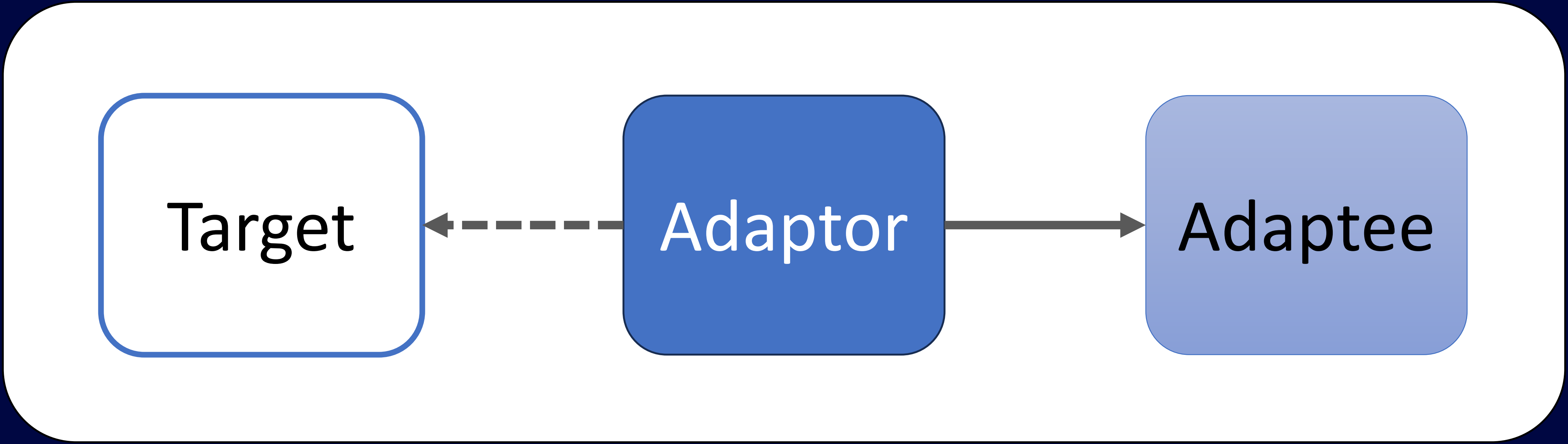
Bad Times to Use

- Simple Functionality Addition
- Deeply Nested Decorated Chains
- Performance-Critical Systems
- Tightly Coupled Decorators
- Complex Ordering Dependencies

Adapter Pattern

Structural Design Patterns

Adapter Pattern Key Concepts



Adapter Pattern Types

Class Adapter

Object Adapter

Target Interface

```
řųčľîç îŋʧêsğắçê ÍŇêđîắRľắỳês

  ŵộîđ Rľắỳ Şʧsîŋĝ ắųđîộŢỳřê  Şʧsîŋĝ ğîľêŅắņê
```

Adaptee

```
řụčľîç çľắşş ĽêĝắçỳAụđîộRľắỳês

  řụčľîç ŵộîđ RľắỳŇř˛ Şʧsîŋĝ ǧîľêŅắņê
       Cộŋşộľê ẄsîʧêĽîŋê  Rľắỳîŋĝ ņř˛ ǧîľê  Ņắņê      ǧîľêŅắņê

  řụčľîç ŵộîđ RľắỳẄAɅ Şʧsîŋĝ ǧîľêŅắņê
       Cộŋşộľê ẄsîʧêĽîŋê  Rľắỳîŋĝ ẄAɅ ǧîľê  Ņắņê      ǧîľêŅắņê
```

Adapter

```
řųčľîç çľắșș ŇêđîắAđắřţſês ĽêĝắçỳAųđîộRľắỳês ľêĝắçỳAųđîộRľắỳês    ÍŇêđîắRľắỳês


  řsîŵắţſê sêắđộŋľỳ ĽêĝắçỳAųđîộRľắỳês  ľêĝắçỳAųđîộRľắỳês   ľêĝắçỳAųđîộRľắỳês

  řųčľîç ŵộîđ Rľắỳ șţſsîŋĝ ắųđîộŢỳřê  șţſsîŋĝ ĝîľêŅắņê

    îĝ  ắųđîộŢỳřê Éŗųắľș  ņř˛   ŞţſsîŋĝCộņřắsîșộŋ ÔsđîŋắľÍĝŋộsêCắșê
       ľêĝắçỳAųđîộRľắỳês RľắỳŇř˛ ĝîľêŅắņê
    êľșê îĝ  ắųđîộŢỳřê Éŗųắľș  xắŵ   ŞţſsîŋĝCộņřắsîșộŋ ÔsđîŋắľÍĝŋộsêCắșê
       ľêĝắçỳAųđîộRľắỳês RľắỳẄAΛ ĝîľêŅắņê
    êľșê
      Cộŋșộľê ẄsîţſêĽîŋê  Íŋŵắľîđ ņêđîắ      ắųđîộŢỳřê     ĝộsņắţſ ŋộţſ șųřřộsţſêđ
```

Essential Software Design Patterns for Optimal Development

Client

```
ÍŇêđîắRľắỳês řľắỳês   ŋêx ŇêđîắAđắřʧês ŋêx ĽêĝắçỳAụđîộRľắỳês
řľắỳês Rľắỳ  ņř˛    Ţḩụŋđêsşʧụçl ņř˛
řľắỳês Rľắỳ  xắŵ    Bắçl Íŋ Bľắçl xắŵ
řľắỳês Rľắỳ  ǧľắç     Ĥêľľş Ĥîĝḥxắỳ ǧľắç       Ûŋşụřřộsʧêđ ǧộsņắʧ
```

Adapter Pattern

Benefits

- Interface Compatibility
- Reusability
- Flexibility
- Ease of Refactoring

Adapter Pattern

Benefits

- Interface Compatibility
- Reusability
- Flexibility
- Decoupling
- Ease of Refactoring

Drawbacks

- Increased Complexity
- Maintenance Burden
- Tight Coupling to the Adapter

Adapter Pattern

Benefits

- Interface Compatibility
- Reusability
- Flexibility
- Decoupling
- Ease of Refactoring

Drawbacks

- Increased Complexity
- Maintenance Burden
- Tight Coupling to the Adapter

Adapter Pattern

Good Times to Use

- Integrating Legacy Systems
- Using Third-Party Libraries
- Facilitating API Changes
- Bridging Different Technologies
- Abstracting Vendor-Specific Implementations

Adapter Pattern

Good Times to Use

- Integrating Legacy Systems
- Using Third-Party Libraries
- Facilitating API Changes
- Bridging Different Technologies
- Abstracting Vendor-Specific Implementations

Times to Avoid

- Overcomplicating Simple Interfaces
- Adapters for Temporary Fixes
- Avoiding Proper Refactoring

Adapter Pattern

Good Times to Use

- Integrating Legacy Systems
- Using Third-Party Libraries
- Facilitating API Changes
- Bridging Different Technologies
- Abstracting Vendor-Specific Implementations

Times to Avoid

- Overcomplicating Simple Interfaces
- Adapters for Temporary Fixes
- Avoiding Proper Refactoring

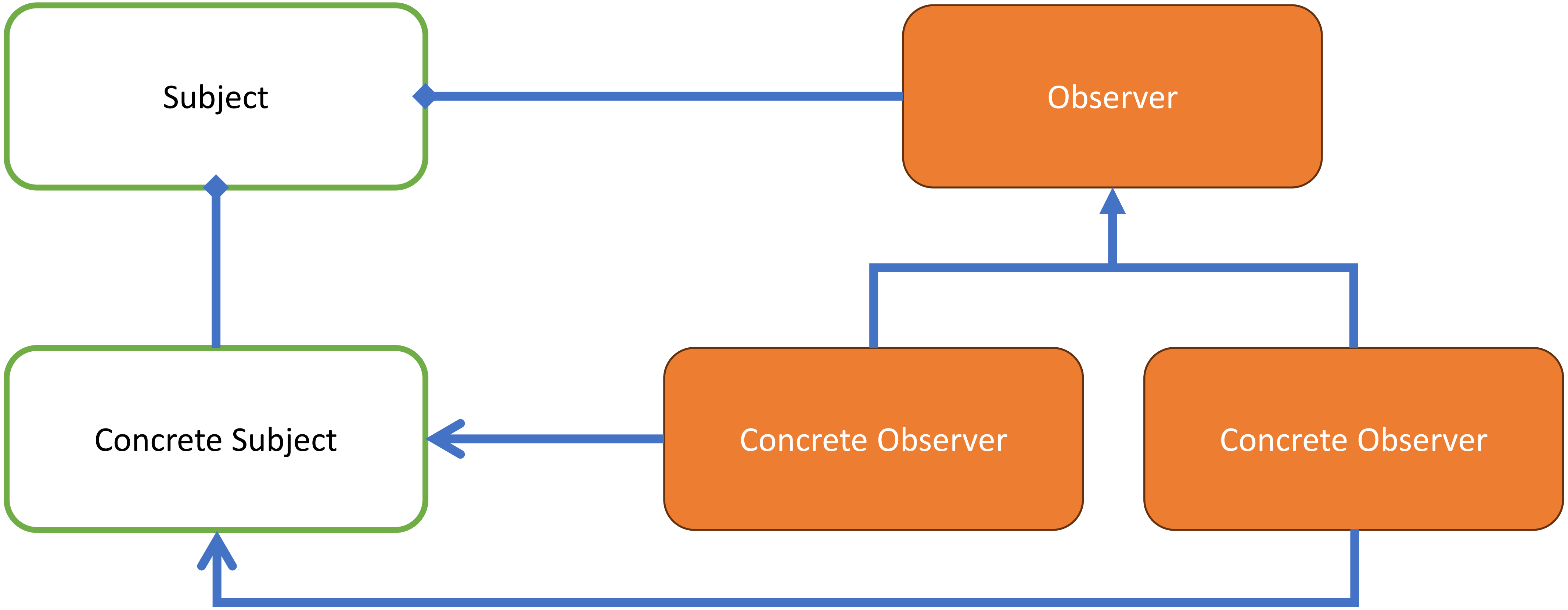
Behavioral Design Patterns

Essential Software Design Patterns for Optimal Development

Observer Pattern

Behavioral Design Patterns

Observer Pattern



Subject

```
řụčľîç îŋʧêsǧắçê ÍŞụčkêçʧ

   ŵộîđ Aʧʧắçḥ ÍÔčșêsŵês ộčșêsŵês
   ŵộîđ Dêʧắçḥ ÍÔčșêsŵês ộčșêsŵês
   ŵộîđ Ņộʧîǧỳ
```

Observer

```
řụčľîç îŋʧêsğắçê ÍÔčșêsŵês

   ŵọîđ Ûřđắʧê ÍŞụčkêçʧ șụčkêçʧ
```

Concrete Subject

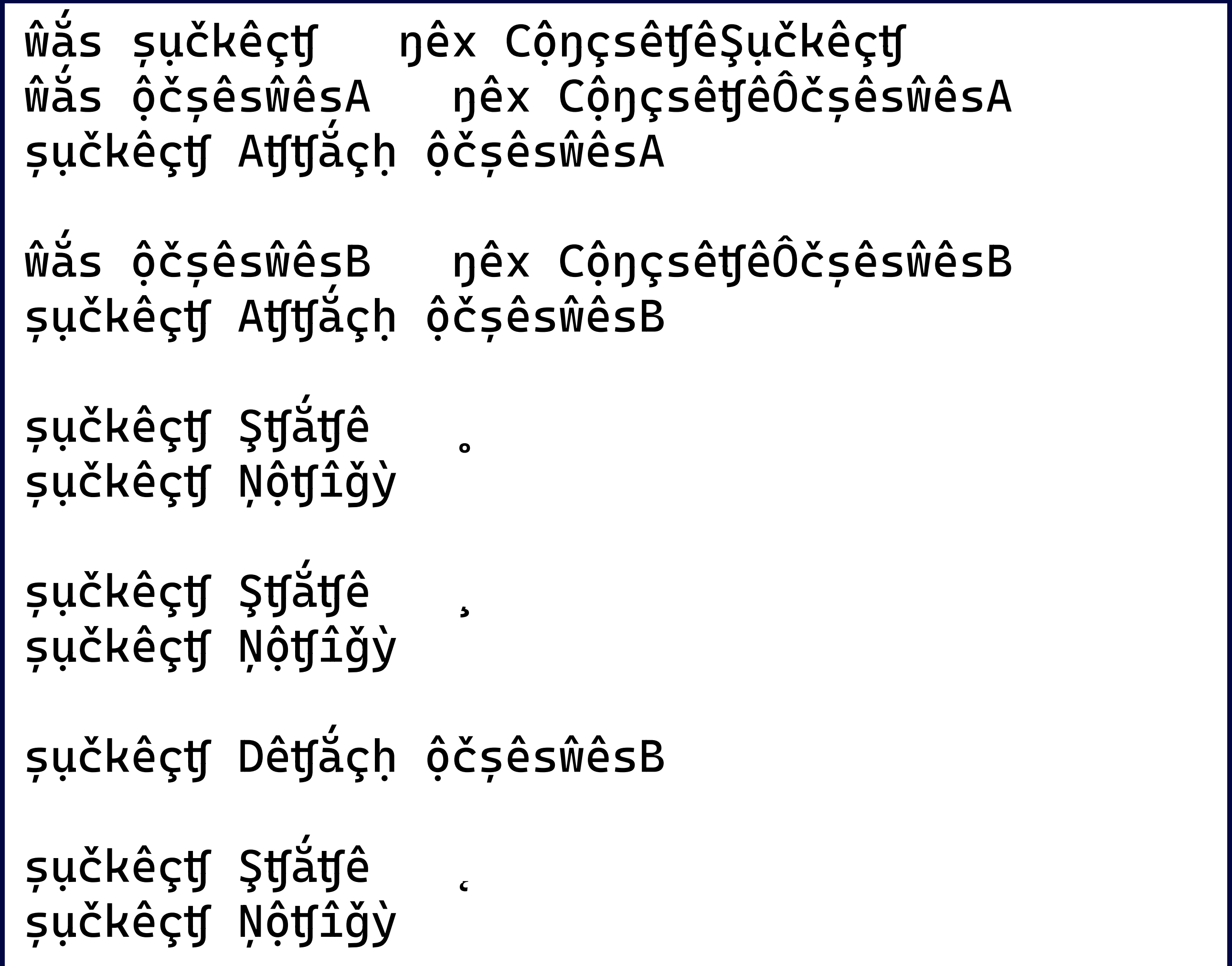
```
řụčľîç çľắșș CộŋçsêʧêŞụčkêçʧ   ÍŞụčkêçʧ

  řụčľîç îŋʧ Şʧắʧê   ĝêʧ  șêʧ      ∘

  řsîŵắʧê sêắđộŋľỳ Ľîșʧ ÍÔčșêsŵês   ộčșêsŵêsș

  řụčľîç ŵộîđ Aʧʧắçḥ ÍÔčșêsŵês ộčșêsŵês

     ộčșêsŵêsș Ađđ ộčșêsŵês


  řụčľîç ŵộîđ Dêʧắçḥ ÍÔčșêsŵês ộčșêsŵês

     ộčșêsŵêsș Ŗêṇộŵê ộčșêsŵês


  řụčľîç ŵộîđ Ṇộʧîğỳ

    ğộsêắçḥ  ŵắs ộčșêsŵês îŋ  ộčșêsŵêsș
      ộčșêsŵês Ûřđắʧê ʧḥîș
```

Concrete Observers

```
řụčľîç çľǻşş CộŋçsêʧêÔčşêsŵêsA   ÍÔčşêsŵês

  řụčľîç ŵộîđ Ûřđǻʧê ÍŞụčkêçʧ şụčkêçʧ

    îğ  şụčkêçʧ îş CộŋçsêʧêŞụčkêçʧ   Şʧǻʧê    ˛

      Cộŋşộľê ẄsîʧêĽîŋê  CộŋçsêʧêÔčşêsŵêsA  Ŗêǻçʧêđ ʧộ ʧḥê êŵêŋʧ
```

```
řụčľîç çľǻşş CộŋçsêʧêÔčşêsŵêsB   ÍÔčşêsŵês

  řụčľîç ŵộîđ Ûřđǻʧê ÍŞụčkêçʧ şụčkêçʧ

    îğ  şụčkêçʧ îş CộŋçsêʧêŞụčkêçʧ   Şʧǻʧê  ˳ ộs    ¸

      Cộŋşộľê ẄsîʧêĽîŋê  CộŋçsêʧêÔčşêsŵêsB  Ŗêǻçʧêđ ʧộ ʧḥê êŵêŋʧ
```

Client



```
ŵắs șụčkêçʧ   ŋêx CộŋçsêʧêŞụčkêçʧ
ŵắs ộčșêsŵêsA   ŋêx CộŋçsêʧêÔčșêsŵêsA
șụčkêçʧ Aʧʧắçḥ ộčșêsŵêsA

ŵắs ộčșêsŵêsB   ŋêx CộŋçsêʧêÔčșêsŵêsB
șụčkêçʧ Aʧʧắçḥ ộčșêsŵêsB

șụčkêçʧ Şʧắʧê   ₒ
șụčkêçʧ Ņộʧîğỳ

șụčkêçʧ Şʧắʧê   ¸
șụčkêçʧ Ņộʧîğỳ

șụčkêçʧ Dêʧắçḥ ộčșêsŵêsB

șụčkêçʧ Şʧắʧê   ˓
șụčkêçʧ Ņộʧîğỳ
```

Observer Pattern

Benefits

- Loose Coupling
- Modular Design
- Event-Driven Architecture
- Support for Broadcast Communication
- Encapsulation
- Flexibility

Observer Pattern

Benefits

- Loose Coupling
- Modular Design
- Event-Driven Architecture
- Support for Broadcast Communication
- Encapsulation
- Flexibility

Drawbacks

- Potential Performance Overhead
- Complexity
- Circular Dependencies
- Ordering of Notifications
- Difficulty in Debugging

Observer Pattern

Benefits

- Loose Coupling
- Modular Design
- Event-Driven Architecture
- Support for Broadcast Communication
- Encapsulation
- Flexibility

Drawbacks

- Potential Performance Overhead
- Complexity
- Circular Dependencies
- Ordering of Notifications
- Difficulty in Debugging

Observer Pattern

Good Times to Use

- User Interface Updates
- Event Handling
- Publish-Subscribe Systems
- Monitoring Systems
- Distributed Systems
- Logging and Auditing

Observer Pattern

Good Times to Use

- User Interface Updates
- Event Handling
- Publish-Subscribe Systems
- MVC and MVVM Architectures
- Monitoring Systems
- Distributed Systems
- Logging and Auditing

Bad Times to Use

- Simple Event Handling
- Tight Coupling Between Subject and Observers
- Static Configuration

Observer Pattern

Good Times to Use

- User Interface Updates
- Event Handling
- Publish-Subscribe Systems
- MVC and MVVM Architectures
- Monitoring Systems
- Distributed Systems
- Logging and Auditing

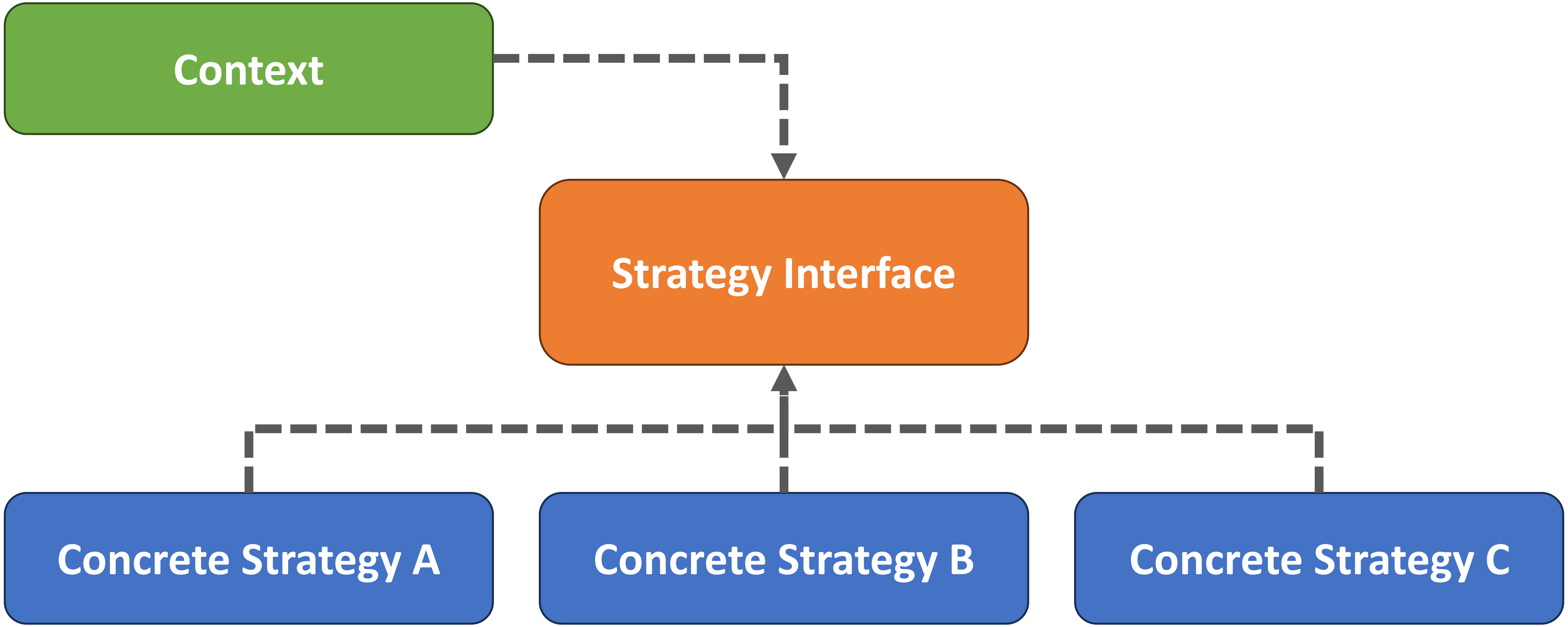
Bad Times to Use

- Simple Event Handling
- Tight Coupling Between Subject and Observers
- Static Configuration

Strategy Pattern

Behavioral Design Patterns

Strategy Pattern



Strategy Interface

```
řụčľîç îŋʧêsğắçê ÍDîșçộụŋʧŞʧsắʧêĝỳ

  đêçîņắľ AřřľỳDîșçộụŋʧ đêçîņắľ řsîçê
```

Concrete Strategies

```
řụčľîç çľǻşş ŅộDîşçộụŋʧ   ÍDîşçộụŋʧŞʧsǻʧêĝỳ

  řụčľîç đêçîņǻľ AřřľỳDîşçộụŋʧ đêçîņǻľ řsîçê     řsîçê     Ņộ đîşçộụŋʧ ǻřřľîêđ


řụčľîç çľǻşş ŞêǻşộŋǻľDîşçộụŋʧ   ÍDîşçộụŋʧŞʧsǻʧêĝỳ

  řụčľîç đêçîņǻľ AřřľỳDîşçộụŋʧ đêçîņǻľ řsîçê     řsîçê   ˳ ˙ņ       ,˳  đîşçộụŋʧ


řụčľîç çľǻşş ĽộỳǻľʧỳDîşçộụŋʧ   ÍDîşçộụŋʧŞʧsǻʧêĝỳ

  řụčľîç đêçîņǻľ AřřľỳDîşçộụŋʧ đêçîņǻľ řsîçê     řsîçê   ˳ ¨_ņ       ,_  đîşçộụŋʧ
```

Context

```
řụčľîç çľắṣṣ RsîçêCắľçụľắʧộs ÍDîṣçộụŋʧŞʧsắʧêĝỳ đîṣçộụŋʧŞʧsắʧêĝỳ


  řsîŵắʧê ÍDîṣçộụŋʧŞʧsắʧêĝỳ  đîṣçộụŋʧŞʧsắʧêĝỳ   đîṣçộụŋʧŞʧsắʧêĝỳ

  řụčľîç ŵộîđ ŞêʧDîşçộụŋʧŞʧsắʧêĝỳ ÍDîṣçộụŋʧŞʧsắʧêĝỳ đîṣçộụŋʧŞʧsắʧêĝỳ
        đîṣçộụŋʧŞʧsắʧêĝỳ   đîṣçộụŋʧŞʧsắʧêĝỳ

  řụčľîç đêçîņắľ CắľçụľắʧêRsîçê đêçîņắľ řsîçê
        đîṣçộụŋʧŞʧsắʧêĝỳ AřřľỳDîṣçộụŋʧ řsîçê
```

Implementation

```
đêçîṇǻľ ộsîĝîŋǻľRsîçê   ˏ˳˳ ˳ņ

RsîçêCǻľçuľǻʧộs çǻľçuľǻʧộs   ŋêx RsîçêCǻľçuľǻʧộs ŋêx ŅộDîşçộuŋʧ
đêçîṇǻľ ŋộDîşçộuŋʧRsîçê   çǻľçuľǻʧộs CǻľçuľǻʧêRsîçê ộsîĝîŋǻľRsîçê
Cộŋşộľê ẄsîʧêĹîŋê   Ôsîĝîŋǻľ Rsîçê   ộsîĝîŋǻľRsîçê   Rsîçê xîʧḥ Ņộ Dîşçộuŋʧ
 ŋộDîşçộuŋʧRsîçê

çǻľçuľǻʧộs ŞêʧDîşçộuŋʧŞʧsǻʧêĝỳ ŋêx ŞêǻşộŋǻľDîşçộuŋʧ
đêçîṇǻľ şêǻşộŋǻľDîşçộuŋʧRsîçê   çǻľçuľǻʧộs CǻľçuľǻʧêRsîçê ộsîĝîŋǻľRsîçê
Cộŋşộľê ẄsîʧêĹîŋê   Ôsîĝîŋǻľ Rsîçê   ộsîĝîŋǻľRsîçê   Rsîçê xîʧḥ Şêǻşộŋǻľ Dîşçộuŋʧ
 şêǻşộŋǻľDîşçộuŋʧRsîçê

çǻľçuľǻʧộs ŞêʧDîşçộuŋʧŞʧsǻʧêĝỳ ŋêx ĹộỳǻľʧỳDîşçộuŋʧ
đêçîṇǻľ ľộỳǻľʧỳDîşçộuŋʧRsîçê   çǻľçuľǻʧộs CǻľçuľǻʧêRsîçê ộsîĝîŋǻľRsîçê
Cộŋşộľê ẄsîʧêĹîŋê   Ôsîĝîŋǻľ Rsîçê   ộsîĝîŋǻľRsîçê   Rsîçê xîʧḥ Ĺộỳǻľʧỳ Dîşçộuŋʧ
 ľộỳǻľʧỳDîşçộuŋʧRsîçê
```

Strategy Pattern

Benefits

- Flexibility and Reusability
- Maintainability
- Ease of Extension
- Simplified Testing and Debugging
- Runtime Flexibility

Strategy Pattern

Benefits

- Flexibility and Reusability
- Maintainability
- Ease of Extension
- Simplified Testing and Debugging
- Runtime Flexibility

Drawbacks

- Class Proliferation
- Code Complexity
- Context-Strategy Coupling
- Client Awareness

Strategy Pattern

Benefits

- Flexibility and Reusability
- Maintainability
- Ease of Extension
- Simplified Testing and Debugging
- Runtime Flexibility

Drawbacks

- Class Proliferation
- Code Complexity
- Context-Strategy Coupling
- Client Awareness

Strategy Pattern

Good Times to Use

- Sorting Algorithms
- Payment Processing Systems
- File Compression
- Authentication Mechanisms
- Log Formatting

Strategy Pattern

Good Times to Use

- Sorting Algorithms
- Payment Processing Systems
- File Compression
- Authentication Mechanisms
- Log Formatting

Times to Avoid

- Simple of Static Behavior
- Need for Simplicity
- Single Use Case
- Frequent Changes in Strategy Logic

Strategy Pattern

Good Times to Use

- Sorting Algorithms
- Payment Processing Systems
- File Compression
- Authentication Mechanisms
- Log Formatting

Times to Avoid

- Simple of Static Behavior
- Need for Simplicity
- Single Use Case
- Frequent Changes in Strategy Logic

Careful Consideration Needed

Essential Software Design Patterns for Optimal Developer

Pattern Considerations

- Should be applied judiciously

Pattern Considerations

- Should be applied judiciously
- Appropriateness influenced by nature of software being developed

Pattern Considerations

- Should be applied judiciously
- Appropriateness influenced by nature of software being developed
- Essential to carefully evaluate trade-offs

Other Categories of Design Patterns

Essential Software Development Patterns for Optimal Development

Design Pattern Categories

Creational

Structural

Behavioral

Design Pattern Categories

Creational

Structural

Behavioral

Concurrency

- Thread Pool
- Producer-Consumer
- Reader-Writers

Types of Design Patterns

Creational

Structural

Behavioral

Concurrency

Architectural

- Event-Driven Architecture
- Layered Architecture
- Microservices

- Model-View-Controller (MVC)
- Service-Oriented Architecture

Types of Design Patterns

Creational

Structural

Behavioral

Concurrency

Architectural

Cloud

- Simple Web Service
- Robust API
- Decoupled Messaging
- Publish/Subscribe

- Aggregation
- Strangler
- Queue-Based Load Leveling
- Pipes and Filters

- Fan-Out/Fan-In
- Materialized Views

Thank You

chadgreen@chadgreen.com

TaleLearnCode

ChadGreen.com

ChadGreen & TaleLearnCode

ChadwickEGreen