Design Patterns in Real Life

Brendan Cassida

Brendan Cassida

Why should you believe me?

- Principal Software Architect for Matrical Biosciences, Spokane, WA
- Architect *and* primary developer for Vitesse and Onda process management platforms for laboratory robotics and VIM compound/biological sample data repository
- 15+ years professional software development experience
  - 10+ years enterprise systems
- Attended EWU for BA/Mathematics and BS/CompSci, 2000-2003
  - EWU ACM President 2002-2003

What is a Design Pattern?

What is a Design Pattern?

A *design pattern* is a reusable, domain-independent strategy for addressing a problem inherent in a system's requirements.

What is a Real Life Design Pattern?

A *real life design pattern* is a reusable, domain-independent strategy for addressing a problem inherent in a system's non-functional requirements.

Design is driven by non-functional requirements.

What is a Non-Functional Requirement?

A *Non-Functional Requirement* indicates a quality of a system, as opposed to a specific behavior.

What is a Non-Functional Requirement?

Simplicity
Responsiveness
Security
Reliability
Portability
Extensibility
Maintainability
Interoperability
Usability
Testability
***ability**

A *Non-Functional Requirement* indicates a quality of a system, as opposed to a specific behavior.

What is a Non-Functional Requirement?

A *Non-Functional Requirement* indicates a quality of a system, as opposed to specific behavior.

Simplicity
Responsiveness
Security
Reliability
Portability
Extensibility
Maintainability
Interoperability
Usability
Testability
***ability**

These are the targets of design patterns.

Know your problem.

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?

Because we want to ensure that there's only one instance of an object in memory.

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?
Because we want to share state across component boundaries.

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?
Because we want to share state across component boundaries.
Share state?
Component boundaries?

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?
Because we want to share state across component boundaries.

Share state? Transferability
Component boundaries? Re-usability

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?
Because we want to share state across component boundaries.
Share state? Transferability
Component boundaries? Re-usability

No shared state?
Static Method

Exempli Gratiā: Singleton

Why would we use the Singleton pattern?
Because we want to ensure that there's only one instance of an object in memory.
Why would we want that?
Because we want to share state across component boundaries.
Shared state? Bad!
Component boundaries? Re-usability.

No shared state?
Static Method
No component boundaries?
Instance Fields

Know your problem.

Solve your problem.

Categorizing Patterns

There is an existing nomenclature for the categorization of patterns.

It is very useful for learning patterns.

Categorizing Patterns

There is an existing nomenclature for the categorization of patterns.

It is very useful for learning patterns.

It is useless for applying patterns in real life.

Your problems are in categorized as *abilities. Your solutions should be categorized the same way.

Know your solution.

Categorizing Patterns

Structural

Creational

Behavioral

Categorizing Patterns

Structural	Clarity Responsiveness Security Interoperability
Creational	Extensibility Responsiveness Re-usability
Behavioral	Extensibility Interoperability Variability

Categorizing Patterns

Structural

Creational

Behavioral

There is a many-to-many relationship between design pattern categories and non-functional requirement categories.

Clarity
Responsiveness
Security
Interoperability

Extensibility
Responsiveness
Re-usability

Extensibility
Interoperability
Variability

Categorizing Patterns

Structural

Creational

Behavioral

Clarity
Responsiveness
Security
Interoperability

Extensibility
Responsiveness
Re-usability

Interoperability
Variability

There is a many-to-many relationship between design pattern categories and non-functional requirement categories.

To understand the applications of real life design patterns, it is necessary to individually re-categorize each design pattern into a set of non-functional requirement categories. *

Categorizing Patterns

Structural

Creational

Behavioral

Clarity
Responsiveness
Security
Interoperability

Extensibility
Responsiveness
Re-usability

Interoperability
Variability

There is a many-to-many relationship between design pattern categories and non-functional requirement categories.

To understand the applications of real life design patterns, it is necessary to individually re-categorize each design pattern into a set of non-functional requirement categories. *

* left as an exercise for the reader

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?

Wrong Answer: To change the structure of our system.

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?
Wrong Answer: To change the structure of our system.
Correct Answer: To increase the responsiveness of a system where many reads occur on a slow-to-load object.

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?
Wrong Answer: To change the structure of our system.
Correct Answer: To increase the responsiveness of a system where many reads occur on a slow-to-load object.
Also Correct Answer: To provide a common API to local and remote collaborators.

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?
Wrong Answer: To change the structure of our system.
Correct Answer: To increase the responsiveness of a system where many reads occur on a slow-to-load object.
Also Correct Answer: To provide a common API to local and remote collaborators.
Also Also Correct Answer: To ensure that an object's methods and states are secure from arbitrary callers.

Exempli Gratiā: Proxy (Structural)

Why would we use the Proxy pattern?
Wrong Answer: To change the structure of our system.
Correct Answer: To increase the responsiveness of a system where many reads occur on a slow-to-load object.
Also Correct Answer: To provide a common API to local and remote collaborators.
Also Also Correct Answer: To ensure that an object's methods and states are secure from arbitrary callers.

Responsiveness
Interoperability
Security

Know your solution.

Apply your solution.

Applying Patterns

When do we begin to consider a design pattern?

Applying Patterns

When do we begin to consider a design pattern?

We consider a design pattern when a non-functional requirement is *added* to or *modified* within our system.

Applying Patterns

When do we begin to consider a design pattern?
We consider a design pattern when a non-functional requirement is *added* to or *modified* within our system.

The correct application of design patterns is dependent on the design methodology and its implications about the evolution of system requirements.

Contrasting Design Methodologies

Traditional Design

Agile Design

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change

Agile Design

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements

Agile Design

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements
- Requirement changes occur at the beginning of a maintenance cycle.

Agile Design

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements
- Requirement changes occur at the beginning of a maintenance cycle.

Agile Design

- Useful for applications, discrete collaborations, systems that must accommodate change

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements
- Requirement changes occur at the beginning of a maintenance cycle.

Agile Design

- Useful for applications, discrete collaborations, systems that must accommodate change
- Implies up-front elicitation of functional requirements only

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements
- Requirement changes occur at the beginning of a maintenance cycle.

Agile Design

- Useful for applications, discrete collaborations, systems that must accommodate change
- Implies up-front elicitation of functional requirements only
- Implies discovery of non-functional requirements

Contrasting Design Methodologies

Traditional Design

- Useful for APIs, frameworks, systems that must resist change
- Implies up-front elicitation of functional and non-functional requirements
- Requirement changes occur at the beginning of a maintenance cycle.

Agile Design

- Useful for applications, discrete collaborations, systems that must accommodate change
- Implies up-front elicitation of functional requirements only
- Implies discovery of non-functional requirements
- Requirements can change at any time

Contrasting Design Methodologies

Traditional Design

- Bulk application of patterns
- Up-front and at infrequent intervals

Agile Design

- Individual application of patterns
- As needed in frequent refactorings

Contrasting Design Methodologies

Traditional Design

- Bulk application of patterns
- Up-front and at infrequent intervals

Agile Design

- Individual application of patterns
- As needed in frequent refactorings
- Exception: Testability

Applying Patterns

When do we begin to consider a design pattern?
We consider a design pattern when a non-functional requirement is *added* to or *modified* within our system.
When do we not consider a design pattern?

Applying Patterns

When do we begin to consider a design pattern?
We consider a design pattern when a non-functional requirement is *added* to or *modified* within our system.
When do we not consider a design pattern?
Whenever we don't have to.

Do not add needless complexity to a system.

Mitigating Complexity

What is complexity?

Mitigating Complexity

What is complexity?

- Complexity is a function of the number of collaborators in a system and the number of collaborations

Mitigating Complexity

What is complexity?

- Complexity is a function of the number of collaborators in a system and the number of collaborations
- Complexity is directly proportional to the product of its variables

Mitigating Complexity

What is complexity?

- Complexity is a function of the number of collaborators in a system and the number of collaborations
- Complexity is directly proportional to the product of its variables

Patterns can increase or decrease complexity.

Know the consequences of your solutions.

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

- Only one instance of an object exists in memory

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

- Only one instance of an object exists in memory
- Global state is exposed

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

- Only one instance of an object exists in memory
- Global state is exposed
- Actions on a Singleton have persistent side effects

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

- Only one instance of an object exists in memory
- Global state is exposed
- Actions on a Singleton have persistent side effects

Transferability
Re-Usability

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?

- Only one instance of an object exists in memory
- Global state is exposed
- Actions on a Singleton have persistent side effects

Transferability
Re-Usability
Complexity
! Testability

Exempli Gratiā: Singleton

What are the consequences of using the Singleton pattern?
Only one instance of an object exists in memory
Global state is exposed
Actions on a Singleton have persistent side effects

Transferability
Re-Usability
Complexity
! Testability

Is this what you really want?

Know the consequences of your solutions.

Conclusion

Know your problem

Conclusion

Know your problem

- Identify non-functional requirements

Conclusion

Know your problem

- Identify non-functional requirements

Know your solution

Conclusion

Know your problem

- Identify non-functional requirements

Know your solution

- Re-categorize patterns

Conclusion

Know your problem

- Identify non-functional requirements

Know your solution

- Re-categorize patterns

Know how and when to apply the solution

Conclusion

Know your problem
- Identify non-functional requirements

Know your solution
- Re-categorize patterns

Know how and when to apply the solution
- Identify design methodology

Conclusion

Know your problem
- Identify non-functional requirements

Know your solution
- Re-categorize patterns

Know how and when to apply the solution
- Identify design methodology
- Mitigate complexity

The End

You may contact me at:

- brendan.cassida@matrical.com
- http://linkedin.com/in/brendancassida
- http://facebook.com/brendan.cassida