“Refactoring” Refactoring

Les Waguespack, Ph.D., Professor
Bentley University

Jeffry Babb, Ph.D. Associate Professor
West Texas A&M University

David J. Yates, Ph.D., Associate Professor
Bentley University

HICSS-50 - Hawaii
January 7, 2017
Inspiration

Understanding design is no less than understanding our relationship as human beings with the world we live in!
Motivation

- Explore refactoring as an evolutionary design activity.
- Extend or “refactor” the concept of code refactoring beyond changes in code structure to improve design quality by incorporating the stakeholders’ experience of the artifact as it relates to their intensions.
- Enhance the clarity, understandability, maintainability, and extensibility manifest in:
  - the stakeholder intentions
  - the artifact
  - their interrelationships
Motivation

- Explore refactoring as an evolutionary design activity.
- Extend or “refactor” the concept of code refactoring beyond changes in code structure to improve design quality by incorporating the stakeholders’ experience of the artifact as it relates to their intensions.
- Enhance the clarity, understandability, maintainability, and extensibility manifest in:
  - the stakeholder intensions
  - the artifact
  - their interrelationships
- Elaborate this “refactored” refactoring as the organizing principle of design as a reflective practice.
Code refactoring is the act of modifying the grammatical structure of source code while retaining the code’s existing behavior; “improving the design of the code after it has been written.” [Fowler et al, 1999]

“Bad Smells” connote a foul situation in need of attention.

Technical debt and Design Debt reflect the expense of “de-coding” poorly, hastily or haphazardly authored program code that results in extraneous effort expended each time the code is attended to (e.g. in maintenance and/or software evolution).
Code Refactoring’s Impact on Design Quality
The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.
The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.

Design quality is expressed through an interpretive framework reflecting the stakeholders’ appreciative system.
The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.

Design quality is expressed through an interpretive framework reflecting the stakeholders’ appreciative system.

The fifteen choice properties of Thriving Systems Theory frame design quality in this study. [Waguespack, 2010]

{derived from Christopher Alexander’s *Nature of Order.*} [Alexander, 2002]
Thriving Systems Properties

- Elegance
- Modularization
- Cohesion
- Composition of Function
- Stepwise Refinement
- Identity
- Encapsulation
- Scale
- Correctness
- User Friendliness
- Reliability
- Transparency
- Programmability
- Patterns
- Extensibility
- Identity
- Elegance
- Modularization
- Cohesion
- Composition of Function
- Stepwise Refinement
- Identity
- Encapsulation
- Scale
- Correctness
- User Friendliness
- Reliability
- Transparency
- Programmability
- Patterns
- Extensibility
- Identity
- Elegance
- Modularization
- Cohesion
- Composition of Function
- Stepwise Refinement
- Identity
- Encapsulation
- Scale
- Correctness
- User Friendliness
- Reliability
- Transparency
- Programmability
- Patterns
- Extensibility
- Identity
- Elegance
- Modularization
- Cohesion
- Composition of Function
Thriving Systems Properties

Stepwise Refinement

Modularization

divisibility

Cohesion

factorability

robustness

Composition of Function

scalability

Encapsulation

Construictability

Correctness

User Friendliness

Identity

Programmability

Transparency

Patterns

Extensibility

Elegance

Stepwise Refinement

“objective” structural (quantifiable)

Correctness

User Friendliness

Identity

Programmability

Transparency

Patterns

Extensibility

Elegance

Stepwise Refinement

“objective” structural (quantifiable)
The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.

Design quality is expressed through an interpretive framework reflecting the stakeholders’ appreciative system.

The fifteen choice properties of Thriving Systems Theory frame design quality in this study. [Waguespack, 2010]

{derived from Christopher Alexander’s *Nature of Order.*} [Alexander, 2002]
The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.

Design quality is expressed through an interpretive framework reflecting the stakeholders’ appreciative system.

The fifteen choice properties of Thriving Systems Theory frame design quality in this study. [Waguespack, 2010]

{derived from Christopher Alexander’s *Nature of Order.*} [Alexander, 2002]

Fowler and Kerievsky propose 80 distinct situations and prescriptions to refactor and remediate “bad smells.”
Code Refactoring’s Impact on Design Quality

- The purpose of code refactoring is to improve the software’s design quality - to increase its “value” without altering its behavior.

- Design quality is expressed through an interpretive framework reflecting the stakeholders’ appreciative system.

- The fifteen choice properties of Thriving Systems Theory frame design quality in this study. [Waguespack, 2010]
  
  {derived from Christopher Alexander’s *Nature of Order.*} [Alexander, 2002]

- Fowler and Kerievsky propose 80 distinct situations and prescriptions to refactor and remediate “bad smells.”

- We identified the predominant TST choice properties attended by those refactoring - and tallied them in the next table . . .

  *Although every choice property plays some role in any design choice, most of the refactorings adhere primarily to an individual property, a few to two and one to three properties.*
# Thriving Systems Choice Properties

<table>
<thead>
<tr>
<th>Structural Properties</th>
<th>Choice Property</th>
<th>Action</th>
<th>Generic Design Action Definition</th>
<th>Refactoring Involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modularization</td>
<td>Modularize</td>
<td>employs or involving a module or modules as the basis of design or construction</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Cohesion</td>
<td>Factor</td>
<td>express as a product of factors</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Encapsulate</td>
<td>enclose the essential features of something succinctly by a protective coating or membrane</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Composition of Function</td>
<td>Assemble</td>
<td>fit together the separate component parts of (a machine or other object)</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Refinement</td>
<td>Elaborate</td>
<td>develop or present (a theory, policy, or system) in detail</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>Focus</td>
<td>(of a person or their eyes) adapt to the prevailing level of light [abstraction] and become able to see clearly</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Identity</td>
<td>Identify</td>
<td>establish or indicate who or what (someone or something) is</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>Pattern</td>
<td>give a regular or intelligible form to</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Programmability</td>
<td>Generalize</td>
<td>make or become more widely or generally applicable</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Friendliness</td>
<td>Accommodate</td>
<td>fit in with the wishes or needs of</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Normalize</td>
<td>make something more normal, which typically means conforming to some regularity or rule</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Correctness</td>
<td>Align</td>
<td>put (things) into correct or appropriate relative positions</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Transparency</td>
<td>Expose</td>
<td>reveal the presence of (a quality or feeling)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td>Extend</td>
<td>render something capable of expansion in scope, effect, or meaning</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Elegance</td>
<td>Coordinate</td>
<td>bring the different elements of (a complex activity or organization) into a relationship that is efficient or harmonious</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Stakeholder Intensions Requirement Elements
Ontological Lens

- Object
- Class
- Remembrance
- Progeny
- Encapsulation
- Variable
- Data attribute
- Method
- Behavioral attribute
- Service
- Behavioral attribute
- Value
- Data attribute
- Identity
- Membership IN
- Message passing
- Relationship
- Association
- Inheritance
- Membership OF
- Ontology Specific Design Actions

Stakeholder
Intensions
Requirement
Elements
Code Refactoring

Ontological Lens

Stakeholder Intensions Requirement Elements

Ontology Specific Design Actions

Target Artifact
Code Refactoring

Ontological Lens

Ontology Specific Design Actions

Stakeholder Intensions Requirement Elements

Design Action Applied Structural Property Influence

Target Artifact

Quality Lens

Structural Properties
Code Refactoring Unpacked
Code Refactoring Unpacked

- **Code refactoring** applies design actions expressible in the ontological scope of the paradigm of construction to rectify deficits of design quality adherent to structural choice properties while preserving the code’s behavior.
Code refactoring applies design actions expressible in the ontological scope of the paradigm of construction to rectify deficits of design quality adherent to structural choice properties while preserving the code’s behavior.

Code refactoring is an exemplar of design (albeit narrowly focused; dedicated as it is to preserving the behavior or functionality already coded):

- a purposeful succession of design actions
- applied to form (or reform) an artifact
- guided by a relevant value proposition
Thriving Systems Properties

Stepwise Refinement  Modularization

Cohesion  Composition of Function

Encapsulation  Scale

User Friendliness  Correctness

Extensibility  Pattern

Reliability  Transparency

Programmability

Identity  Elegance
Thriving Systems Properties

- Elegance
- Divisibility
- Robustness
- Scalability
- Construability
- Transparency
- Programmability
- Reliability
- Patterns
- Extensibility
- User Friendliness
- Correctness
- Composition of Function
- Scale
- Cohesion
- Factorability

“objective” structural (quantifiable)

Stepwise Refinement
Modularization
Thriving Systems Properties

“subjective” aesthetic (experiential)

“objective” structural (quantifiable)

Stepwise Refinement
Modularization
Composition of Function
Scale
Correctness
Identity
usability
predictability
Patterns
Extensibility
User Friendliness
fidelity
correctness
confidence
thriving

Elegance
intuitiveness
effectiveness
user friendliness
Predictability
Sustainability
Vitality
Robustness
Factorability
Construtability
Factorability
Constructability
Removal
Composition of Function
Scale
Correctness
Identity
Elegance
intuitiveness
effectiveness
user friendliness
Predictability
Sustainability
Vitality
Robustness
Factorability
Construtability
Factorability
Constructability
Removal
Code Refactoring

Ontological Lens
- Class
- Instance
- Inheritance
- Polymorphism
- Message passing
- Association
- Remembrance
- Variable
- Property
- Progeny
- Encapsulation
- Identity
- Membership IN
- Membership OF

Quality Lens
- Stakeholder
- Intensions
- Requirement
- Elements
- Design action applied
- Structural property influence

Structural Properties
- Ontology Specific Design Actions
- Constructibility
- Factorability
- Divisibility
- Robustness
- Scalability
- Stepwise Refinement
- Modularization
- Cohesion
- Encapsulation
- Composition of Function
- Scale

Target Artifact
Refactoring “refactored as” Reflection-Driven Design

Ontological Lens

Stakeholder intensions
Requirement elements

Design action applied
Structural property influence
Aesthetic property influence

Quality Lens

Structural properties

Ontology specific design actions

Target artifact

©LJW, 2017 All Rights Reserved
The Result of Generalization
The Result of Generalization

- **Code refactoring** applies design actions expressible in the ontological scope of the language paradigm to rectify deficits of design quality adherent to structural choice properties while preserving the code’s behavior.

BECOMES

- **Design refactoring** applies design actions expressible in the ontological scope of the language paradigm to either of or both the artifact as rendered and the stakeholders’ intensions to rectify deficits of design quality adherent to structural, behavioral, and aesthetic design concerns.
Reflection-Driven Design

Recipe

A: (re-)form an expression of intent
B: specify an ontology for the design space
C: define a collective appreciative system
D: align choice properties and design actions
E: (re-)construct the artifact
F: reflect on the alignment of artifact with intent
G: if mis-aligned repeat (D)-(E)-(F)-(G)
H: if aligned but unsatisfactory restart at (A)
I: resume at (F) as needed for the artifact’s “life”
Code refactoring (ala Fowler & Kerievsky) reflects a design space of object orientation.
Summary Notes

- Code refactoring (*ala Fowler & Kerievsky*) reflects a design space of object orientation.

- A reflective conversation refactoring the wholeness of an artifact interprets the choice properties of the appreciative system through ontology-specific design actions.
Summary Notes

- Code refactoring (ala Fowler & Kerievsky) reflects a design space of object orientation.
- A reflective conversation refactoring the wholeness of an artifact interprets the choice properties of the appreciative system through ontology-specific design actions.
- Ongoing work includes special ontologies for entity relationship modeling and IS security to extend refactoring to those domains.
Summary Notes

- Code refactoring (*ala Fowler & Kerievsky*) reflects a design space of object orientation.

- A reflective conversation refactoring the wholeness of an artifact interprets the choice properties of the appreciative system through ontology-specific design actions.

- Ongoing work includes special ontologies for entity relationship modeling and IS security to extend *refactoring* to those domains.

- The special ontology(s) determine the range of possible artifacts that may exist while the appreciative system (e.g. TST) explain “why” the stakeholders value the artifact.
Code refactoring (*ala Fowler & Kerievsky*) reflects a design space of object orientation.

A reflective conversation refactoring the wholeness of an artifact interprets the choice properties of the appreciative system through ontology-specific design actions.

Ongoing work includes special ontologies for entity relationship modeling and IS security to extend *refactoring* to those domains.

The special ontology(s) determine the range of possible artifacts that may exist while the appreciative system (e.g. TST) explain “why” the stakeholders value the artifact.

Integrating the structural, behavior, and aesthetic aspects of design quality is a natural *refactoring of design* that extends the iterative cycle of construction and theory building to broader concerns to enrich both design practice and design pedagogy.
What do you think?
What do you think?

It is difficult to conceive that code refactoring, even if it indwells principally within structural aspects of a design, will not spill over into a broader spectrum of design concerns.

Agile methods are particularly “tuned in” to this possibility, we believe there are broader opportunities to “refactor” our own discourse regarding the role code refactoring plays in the overall design of systems.

Specifically, we advocate for an empowered developer who, in her reflection-in-action and -on-action, often engages in double-loop learning to refactor stakeholder intensions, and subsequently engage stakeholders in iterative design.
discussion

LWaguespack@Bentley.edu  JBabb@WTAMU.edu  DYates@Bentley.edu