An Introduction to Residuality Theory

Barry M O’Reilly
Black Tulip Technology
Barry O’Reilly

• Founder at Black Tulip Technology
• PhD Candidate Complexity Science & Software Engineering
• Startup CTO
• CITA-P Certified Architect
• Chief Architect at Microsoft Consulting Western Europe
• Worldwide Lead Solutions Architecture Community, Microsoft
Cloud Architecture

- Cloud is big and scary
- Release cadence
- Fast moving ideas
- Constant change
- Business critical
- Many feel intimidated
- How do I keep up with architecting in the cloud?
- The old ways of learning will leave you burnt out very quickly
What is Cloud?

• It’s not hosting
• It’s not a business model
• It’s an entirely new architecture
• Service orientation applied to:
  • Hardware
  • Network
  • Software components
  • Software packages
Cloud Architecture

• Becoming a cloud architect is hard
• It’s not about learning vendor tools
• Old style architecture won’t help
• You need to think differently
• You need to learn differently
• Old silos like infrastructure, security, and development are blurred
• Huge uncertainty
  • Domain
  • Technology
  • Process
• As technology becomes more complicated, the world becomes more complex and interconnected
• Developers are expected to learn cloud and architecture at the same time.
The Mission

• Create cloud architects quickly
• Give a good base in modern software architecture
• New, relevant tools
• Redefine architecture as a science of uncertainty
Antifragile System Design

- A set of heuristics for designing systems in complex environments
- Combines complexity science & software engineering
- Very popular!
- Work now is focused on bringing rigour to this
- Producing science instead of hype
Residuality Theory

An Introduction to Residuality Theory: Software Design Heuristics for Complex Systems
Barry M O’Reilly

Residuality Theory: Proactive Risk Management in the Design Phase
By Barry M. O’Reilly

"There Is No Spoon": Residuality Theory & Rethinking Software Engineering
By Barry M. O’Reilly

The Philosophy of Residuality Theory
Barry M O’Reilly

The 12th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 23-26, 2021, Warsaw, Poland
Architecture as Science

• Residuality Theory provides a means for architects to understand their profession

• A scientific ground that allows us to assess, assimilate, and understand different frameworks, tools, and technologies.

• Peer reviewed and scientifically verified - not another fad with a book and a pretty animal on the front
Philosophy of Architecture
What is Architecture?
Component Metaphor
Architecture and Cloud

- Is software architecture ready for cloud?
- Are current architecture practices even fit for purpose?
  - MBA-lite
  - Alienates developers
  - Terribly boring
  - Focused on internal IT-department politics
  - Command and control at the centre
Design

• Predicting the future
• Based on requirements
• Focused on process and component identification
• Fast abstractions
• Neglected discipline
Risk Management

• Focused on project risk
• Siloed
• Based on predicting the future
• Ignores compound risk
• More comfort than competence
Architecture in Disarray

- Requirements is broken
- Risk management is broken
- Design is broken
- Engineering believes in cults now
- The tools and the thinking are the same as early 2000’s
  - Planning and control at the centre
- We need something else for cloud
Residuality Theory

• Systems in complex environments cannot be modelled as processes and components
• These systems can be described as a stack of residues
• A residue is a fuzzy view of the system under a particular form of stress
• A residue is a set consisting of people, flows of information, and software functions.
Residues
Mathematical Modeling of Complex Systems

Fig. 11. Information transmission through $q$-connected simplices,

(a) 3-connected 6-dimensional simplices  
(b) $q$-transmission dynamics

\[
\delta_{12}f\sigma_1 \Rightarrow \delta_{12}f\sigma_2
\]

(a) 1-transmission from $\sigma_1$ to $\sigma_2$  
(b) 1-transmission along a chain of 1-connection
Builds on complexity science
Assumes Fat tailed distributions and non-predictability
Rejecting Prediction and control
Uses stress as the driver of design decisions

- Stress allows us to move away from requirements or changes
- Allows stakeholders to answer in narratives, not definites
- Captures negative stories
- (Remember no probability)
- Captures realistic risk picture
Designing with Residuality Theory

• This radically changes how we think about design
• We design by carrying out a residual analysis
• All the old school techniques disappear. We don’t need them anymore.
Non-functional Requirements

• This method solves the ever-present issue of ignored non-functional requirements
• It solves it in a way that is directly linked to engineering - there is a result, not just a meeting and a spreadsheet
• Forces engineers to take this seriously as a first step in the design process.
Residual Analysis

- Residual Analysis is the application of residuality theory to the process of software design
- It consists of two major parts
  - Stressor analysis
  - Contagion analysis
Stressor Analysis

• Sensemaking - the act of understanding a complex system through narrative
• But we force the narrator to focus on the negative
  • Because the positive is always kataphatic
• Before we start, we should have a basic idea of what we are trying to achieve - the naïve architecture.
• As an architect you need serious background here
  • The organization
  • The market
  • The society
Stressor analysis

- Just make a list of stressors
- Take half of the list
- Sketch the residues
- Add as much detail as you need
- Use many sources
- NO probability. No discussions about probability.
<table>
<thead>
<tr>
<th>Stressor #</th>
<th>Stressor</th>
<th>Impacts the ability to</th>
<th>Detection</th>
<th>Mitigation</th>
<th>Technical Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New car models</td>
<td>Charge cars</td>
<td>Industry contacts</td>
<td>Flexible charging connectors, adaptors, extra space for new tech</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Electricity failure</td>
<td>Charge cars</td>
<td>Alarm</td>
<td>Batteries</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Failed login</td>
<td>Charge for customer, identify customer</td>
<td>Software alert</td>
<td>Allow charge, invoice later on registration</td>
<td>ALPR</td>
</tr>
<tr>
<td>4</td>
<td>Electric car market crashes</td>
<td>Make money</td>
<td>Stock prices, consumer attitudes</td>
<td>Convert to petrol stations</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Queues</td>
<td>Charge cars</td>
<td>Cameras, sensors</td>
<td>Capacity planning, extra space for expansion</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Competitors cheaper</td>
<td>Attract customers</td>
<td>Market scanning</td>
<td>Over value add in stores</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Competitors lockin/manufacturer</td>
<td>Attract customers</td>
<td>Market scanning</td>
<td>Lobbying, information spreading, partnership</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Damage (accident/criminal)</td>
<td>Charge cars</td>
<td>Sensors</td>
<td>Redundancy, security cameras</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Server failure</td>
<td>Customers to unlock cars</td>
<td>Software alert</td>
<td>Change unlock mechanism.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Billing errors</td>
<td>attract customers, customer loyalty</td>
<td>complaints</td>
<td>waive fees</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Competitors better coverage</td>
<td>Attract customers</td>
<td>Market scanning</td>
<td>Increase investment</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Customer abandons car in slot</td>
<td>Charge cars</td>
<td>ALPR</td>
<td>Invoice per minute</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Privacy regulation</td>
<td>Store customer data, attract customers</td>
<td>Legal department investigation</td>
<td>Different solutions for different geographies</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ICE-ing</td>
<td></td>
<td></td>
<td>Decouple at country level</td>
<td></td>
</tr>
</tbody>
</table>

* 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

**Example**

- **Stressor #**
- **Stressor**
- **Impacts the ability to**
- **Detection**
- **Mitigation**
- **Technical Mitigation**
Properties & patterns

- Modularity, weak links, redundancy, diversity.
- These make resilient systems
- Which combination though?
- Every system is different
- No patterns or best practices
- Residuality theory helps us identify the right combination of these properties
Bagging and Boosting

- Use a training set and a testing set
- Prove that we are better off
  - If the residual system survives stress in the testing set, we have empirical evidence that it survives unknown stress
- Change the order of the stressors and run the process again
- Compare the architectures generated
- Eventually we will have computer support for this
Stressor Analysis

- Eventually it gets hard to find new stressors
- Now we have designed a system for every stressor
- Lots of residues
Contagion analysis

• Now we start to investigate what the system should look like
• We start integrating the residues
• Deciding how to boil the residues down into as few components as possible, whilst retaining their ability to survive the stressors
• We need to prevent contagion when we integrate - one stressor cannot cross a residual boundary and cause issues for another residue’s ability to meet stress
• Here we will produce a list of trade-off decisions to be made.
Tools

• We use Design Structure Matrices to investigate dependencies across flows and functions.

• We use incidence matrices to find which competent are affected by similar types of stress - which identifies residues that can be integrated quickly, where the trade-offs are, and where the outliers are.

• We add component boundaries where we need to prevent contagion, and also where we feel like (old heuristics die hard)
## Contagion Analysis: Stressor-Flow-Component

<table>
<thead>
<tr>
<th>Flows</th>
<th>RegisterCustomer</th>
<th>ReceiveOrder</th>
<th>VerifyOrder</th>
<th>ShipOrder</th>
<th>CancelOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterCustomer</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReceiveOrder</td>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifyOrder</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShipOrder</td>
<td>Stressors/Flows</td>
<td>RegisterCustomer</td>
<td>ReceiveOrder</td>
<td>VerifyOrder</td>
<td>ShipOrder</td>
</tr>
<tr>
<td>CancelOrder</td>
<td>GDPR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CustomerInfoInvalid</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CheckGDPRConsent</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateCustomer</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConfirmRegistration</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CancelRegistration</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### New products

<table>
<thead>
<tr>
<th>New product</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
</table>

### New customer segment (businesses)

<table>
<thead>
<tr>
<th>New customer segment (businesses)</th>
<th>1</th>
</tr>
</thead>
</table>

### Total

<table>
<thead>
<tr>
<th>Total</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

## Stressor/Component

<table>
<thead>
<tr>
<th>Stressor/Component</th>
<th>RegisterCustomerComponents Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDPR Changes</td>
<td>1</td>
</tr>
<tr>
<td>Customer Registration Process Changes</td>
<td>1</td>
</tr>
<tr>
<td>Customer Validation rules change</td>
<td>1</td>
</tr>
</tbody>
</table>

### Component change rate

<table>
<thead>
<tr>
<th>Component change rate</th>
<th>3</th>
</tr>
</thead>
</table>

### Orchestrator

<table>
<thead>
<tr>
<th>Orchestrator</th>
<th>ValidateCustomerInfo</th>
<th>CustomerInfoInvalid</th>
<th>CheckGDPRConsent</th>
<th>CreateCustomer</th>
<th>ConfirmRegistration</th>
<th>CancelRegistration</th>
<th>Components Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterCustomer</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ReceiveOrder</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>VerifyOrder</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ShipOrder</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>CancelOrder</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

### GDPR Changes

<table>
<thead>
<tr>
<th>GDPR Changes</th>
<th>1</th>
</tr>
</thead>
</table>

### Customer Registration Process Changes

<table>
<thead>
<tr>
<th>Customer Registration Process Changes</th>
<th>1</th>
</tr>
</thead>
</table>

### Customer Validation rules change

<table>
<thead>
<tr>
<th>Customer Validation rules change</th>
<th>1</th>
</tr>
</thead>
</table>

### Component change rate

<table>
<thead>
<tr>
<th>Component change rate</th>
<th>3</th>
</tr>
</thead>
</table>
Systems Engineering

• DevOps
• FMEA
• ATAM
• Testing strategies
• Scale units
• Telemetry plans
Conclusion

• Residuality theory provides a scientific basis for practicing architecture
• It allows us to teach and train architects quickly
• It allows us to learn cloud quickly
• It allows us to produce solid cloud architectures that have a chance of surviving in today’s unpredictable world
• It allows us to assess and understand other architectural practices and how effective they might be
The Course

• 4 intense days
• Working with designing a cloud architecture
• Learning how to learn.
• Day 1: Introduction to complexity, antifragility, and residuality
• Day 2: Learn Azure/AWS in one day
• Day 3: Design a cloud system
• Day 4: Design testing, politics, and evolution.
Residual Analysis Recap

• We arrive at a solid design
  • Survives known and unknown stress - real resilience
  • Has all required functionality
    • Any functionality not discussed in the stressor analysis is simple.
  • Exposed holes in the business thinking

• We’ve avoided
  • Requirements engineering
  • Making impossible predictions about the future
  • Linear risk management
  • Missing non-functionals
  • Misunderstanding trade-offs
Residual Analysis

• This replaces the need for any form of architectural framework
  • It covers everything.
  • Bye bye TOGAF!

• Performing the residual analysis is also the toughest form of architect training - it makes apprenticeship possible instead of courses about frameworks or methods.

• It’s also tough - requires real critical thinking skills.

• The entire process reinforces heuristics - you will get faster and faster at this
Getting Started

• Read the articles
• Try the techniques
• Build from there
Contact

barry@blacktulip.se

• LinkedIn: https://www.linkedin.com/in/barry-o-reilly-b924657/