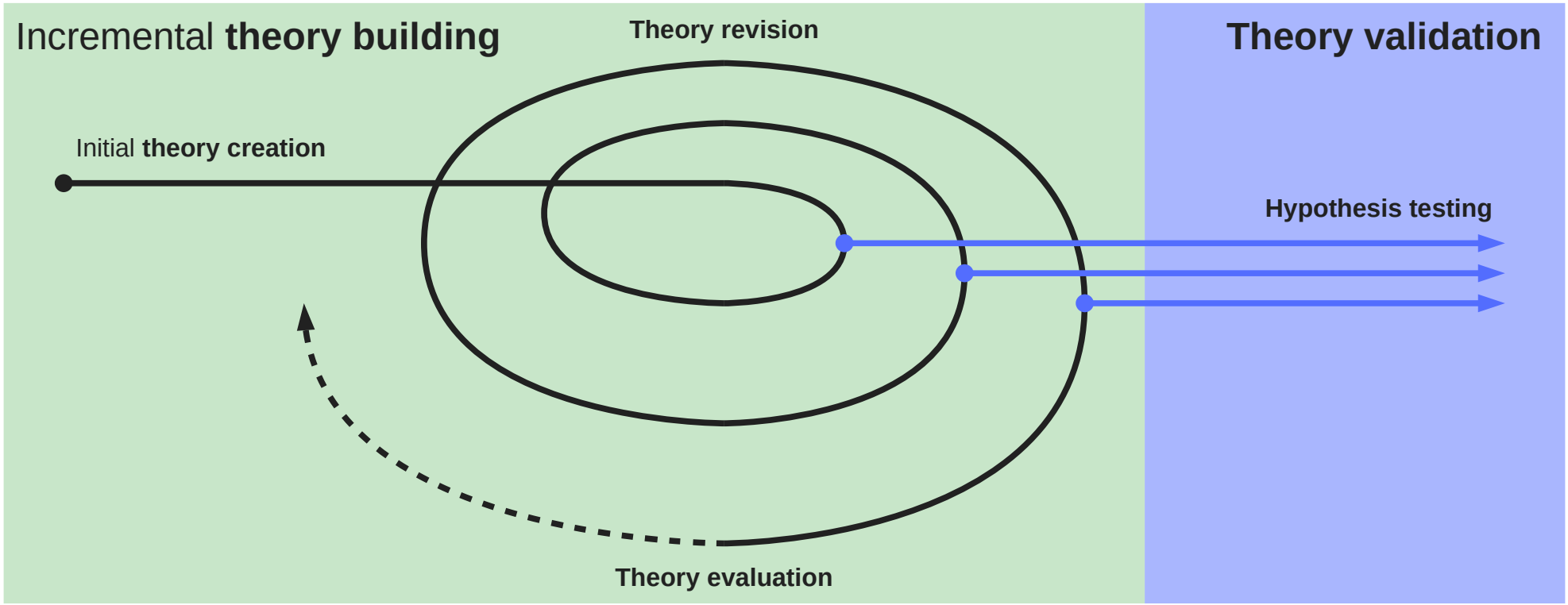
Theory Presentation Using the Handbook Method

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

2021-03-23 Dagstuhl Seminar

Theory Building and Validation



Not drawn to scale or effort involved

[1] See https://dirkriehle.com/2020/02/21/theory-building-and-validation-in-computer-science/

What is a Theory?

Definitions

- Theory
  - Knowledge to be used for correct prediction (and reliable outcome)
- Science
  - The process of building and validating theories

Maxwell’s Equations [1]

Name	Integral equations	Differential equations
Gauss's law	$\oiint_{\partial\Omega} \mathbf{E}\cdot\mathrm{d}\mathbf{S} = \frac{1}{\varepsilon_0}\iiint_{\Omega}\rho\,\mathrm{d}V$	$\nabla\cdot\mathbf{E} = \frac{\rho}{\varepsilon_0}$
Gauss's law for magnetism	$\oiint_{\partial\Omega} \mathbf{B}\cdot\mathrm{d}\mathbf{S} = 0$	$\nabla\cdot\mathbf{B} = 0$
Maxwell–Faraday equation (Faraday's law of induction)	$\oint_{\partial\Sigma} \mathbf{E}\cdot\mathrm{d}\boldsymbol{\ell} = -\frac{\mathrm{d}}{\mathrm{d}t}\iint_{\Sigma}\mathbf{B}\cdot\mathrm{d}\mathbf{S}$	$\nabla\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}$
Ampère's circuital law (with Maxwell's addition)	$\oint_{\partial\Sigma} \mathbf{B}\cdot\mathrm{d}\boldsymbol{\ell} = \mu_0\left(\iint_{\Sigma}\mathbf{J}\cdot\mathrm{d}\mathbf{S} + \varepsilon_0\frac{\mathrm{d}}{\mathrm{d}t}\iint_{\Sigma}\mathbf{E}\cdot\mathrm{d}\mathbf{S}\right)$	$\nabla\times\mathbf{B} = \mu_0\left(\mathbf{J} + \varepsilon_0\frac{\partial\mathbf{E}}{\partial t}\right)$

[1] See https://en.wikipedia.org/wiki/Maxwell%27s_equations

Theory Presentation using Prose [1]

companies. Figure 1 shows problems with organizational structure, and Figure 2 shows how these problems affect the domain engineering process. Both figures share some of the same causes; they have been split up for readability purposes.

The key result is the following:

A root cause, the separation of product units as profit centers from a platform organization as a cost center, leads to delayed deliveries, increased defect rate, and redundant software components.

In our case studies, the business unit owns all products and the platform, a product unit develops a particular product, and a platform organization supports the product units in their work by providing shared reusable assets. The business units in our case studies are all structured into product units as profit centers and the platform organization as a cost center.

As Figure 1 shows, the problems encountered with the organizational structure are traced back to the "separation of product units as profit centers and platform organization as cost center", which makes them "silos" in the language of our interview partners, that is, organizational units that do not collaborate sufficiently. Specifically, the "separation of product units as profit centers" leads to

- "lack of global business unit perspective" where each product unit acts in their own interests irrespective of possible synergies from collaboration,
- "insufficient trust between product units" where other product units are viewed as threats or competitors rather than possible collaborators,
- "power play between product units" where managers in some or all of the product units are fighting to enforce their interests irrespective of other product unit needs,
- "insufficient developer networking" where developers do not find the time to talk to each other across the organizational unit boundaries.

In addition, the separation starves the platform organization for resources. This leads to

- "lack of resources at platform organization", because profit centers responsible for their own revenue are always in a stronger position to hire developers than any cost center.

Figures 1 and 2 do not show every cause and effect relationship, and discussing all interactions is beyond a reasonable length for this article. In the following subsections, we therefore focus on the following three central cause-and-effect chains:

1. Lack of resources at platform organization → Delayed domain artifact realization → Delayed product delivery
2. Power play between product units → Poorly prioritized domain requirements → Rework and wasted effort → Delayed product delivery
3. Insufficient intra-organizational-unit collaboration → Limited understanding of other organizational units → Unclear reusable assets requirements → Insufficient reusable asset quality → Increased defect rate

We chose these chains for presentation because they received the most mentions and interest in our interviews.

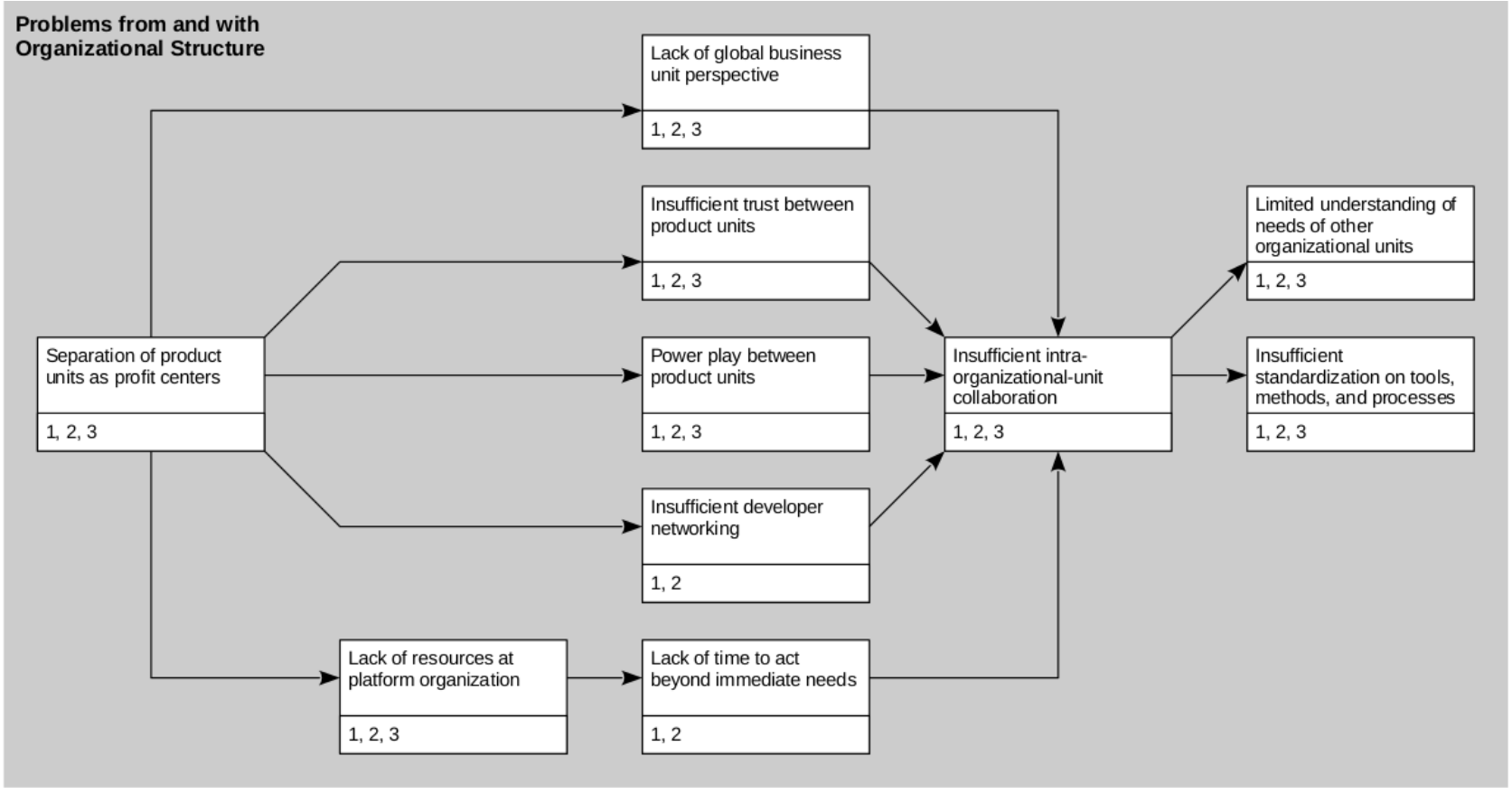
4.2.1 Chain 1: Lack of resources

In all three cases, the platform organization had a significantly higher workload than any of the product units, despite the more direct pressure on the product unit to deliver a product.

"All this work overload leads to lower code quality. I just finish up as quickly as I can and then move on." Developer (platform), case 1.

[1] See Riehle, D., Capraro, M., Kips, D., & Horn, L. (2016). Inner Source in Platform-Based Product Engineering. IEEE Transactions on Software Engineering vol. 42, no. 12 (December 2016), 1162-1177.

Theory Presentation Using Cause-Effect Diagrams



Theory Presentation Using Hypotheses

5.2.2 Hypotheses and Predictions

Our case study companies have been continuing their efforts. However, it is too early to tell whether our recommendations have been beneficial to them.

The theories we present are only as good as the hypotheses that they generate and that can be validated in future work. Such confirmatory research will also allow for generalized conclusions that are not possible from pure case study research.

H1 Resistance and misunderstandings (like expected lower code quality of inner source components) can be addressed successfully by way of education and active participation in the practice of inner source software development.

This hypothesis is likely to evaluate to true, given the change in public opinion on the use of and participation in open source software projects from a negative to a positive stance.

H2 Psychological openness or resistance to inner source (i.e. desire or fear to work under quasi-public scrutiny) depends on manager and developer personalities and is not a function of organizational structure or process.

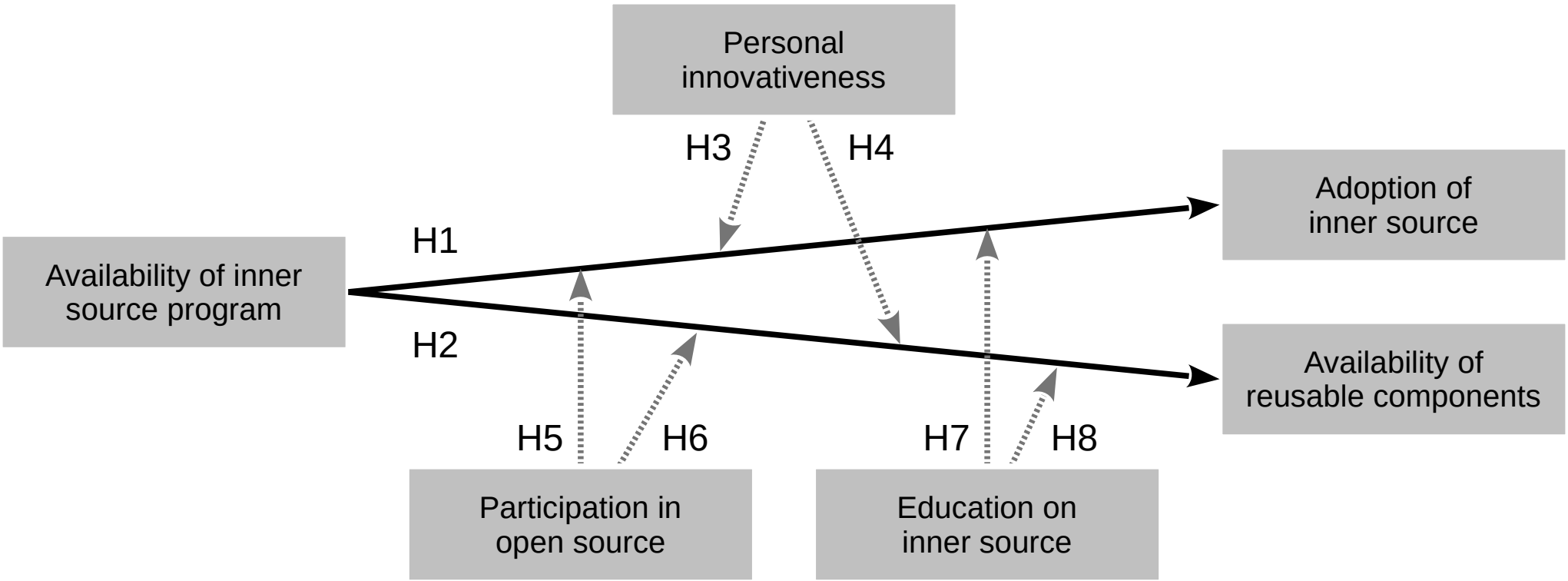
...

Thus, we suggest that open source and inner source competencies are structurally similar, if not isomorphic. This is not surprising given that inner source has originally been motivated by open source. This hypothesized relationship then leads to our most potent hypothesis:

H5 While there is no doubt about the need of platform software and shared reusable assets, a platform development organization may not be needed any longer. It can be replaced by an inner source program.

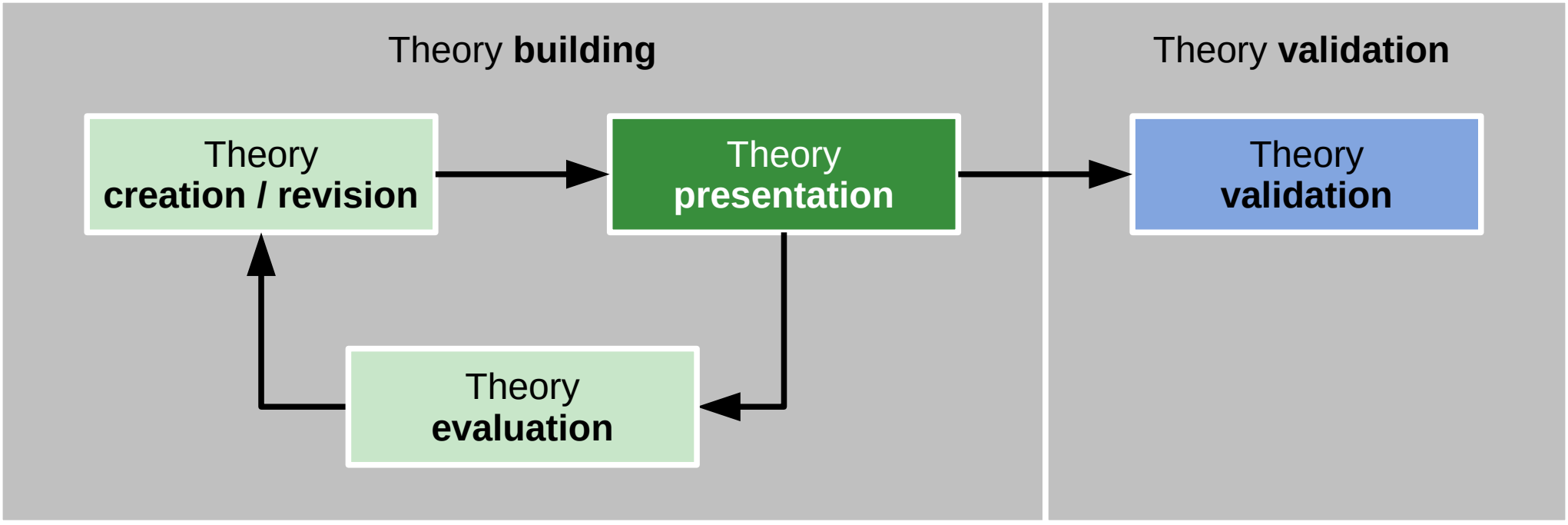
This is an interesting though probably controversial hypothesis: If large companies can work together in an open source foundation to develop shared infrastructure components, why can't product units within an organization work together to create a platform of shared reusable assets without the need for a dedicated organizational unit that maintains this platform?

Theory Presentation Using Research Models



Purpose of Theory Presentation

Theory Presentation in the Scientific Process



The Next Step After Theory Presentation

- **Theory evaluation**
  - Action research
  - Case study research
  - ...
- **Theory validation**
  - Controlled experiments
  - Hypothesis testing surveys
  - ...

Example Research Design for Microservices Integration Theory

1. Theory creation
   - **Qualitative survey (→ theory)**
2. Theory evaluation and revision
   - **Action research (→ revised theory)**
3. Theory evaluation
   - **Multiple-case case study research**

One More Thing...

companies. Figure 1 shows problems with organizational structure, and Figure 2 shows how these problems affect the domain engineering process. Both figures share some of the same causes; they have been split up for readability purposes.

The key result is the following:

A root cause, the separation of product units as profit centers from a platform organization as a cost center, leads to delayed deliveries, increased defect rate, and redundant software components.

In our case studies, the business unit owns all products and the platform, a product unit develops a particular product, and a platform organization supports the product units in their work by providing shared reusable assets. The business units in our case studies are all structured into product units as profit centers and the platform organization as a cost center.

As Figure 1 shows, the problems encountered with the organizational structure are traced back to the "separation of product units as profit centers and platform organization as cost center", which makes them "silos" in the language of our interview partners, that is, organizational units that do not collaborate sufficiently. Specifically, the "separation of product units as profit centers" leads to

- "lack of global business unit perspective" where each product unit acts in their own interests irrespective of possible synergies from collaboration,
- "insufficient trust between product units" where other product units are viewed as threats or competitors rather than possible collaborators,
- "power play between product units" where managers in some or all of the product units are fighting to enforce their interests irrespective of other product unit needs,
- "insufficient developer networking" where developers do not find the time to talk to each other across the organizational unit boundaries.

In addition, the separation starves the platform organization for resources. This leads to

- "lack of resources at platform organization", because profit centers responsible for their own revenue are always in a stronger position to hire developers than any cost center.

Figures 1 and 2 do not show every cause and effect relationship, and discussing all interactions is beyond a reasonable length for this article. In the following subsections, we therefore focus on the following three central cause-and-effect chains:

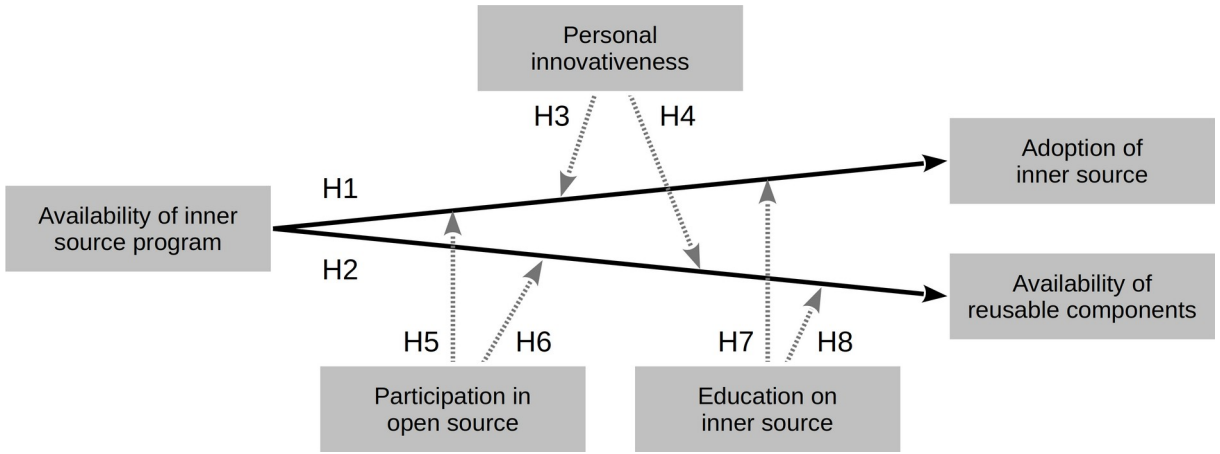
1. Lack of resources at platform organization → Delayed domain artifact realization → Delayed product delivery
2. Power play between product units → Poorly prioritized domain requirements → Rework and wasted effort → Delayed product delivery
3. Insufficient intra-organizational-unit collaboration → Limited understanding of other organizational units → Unclear reusable assets requirements → Insufficient reusable asset quality → Increased defect rate

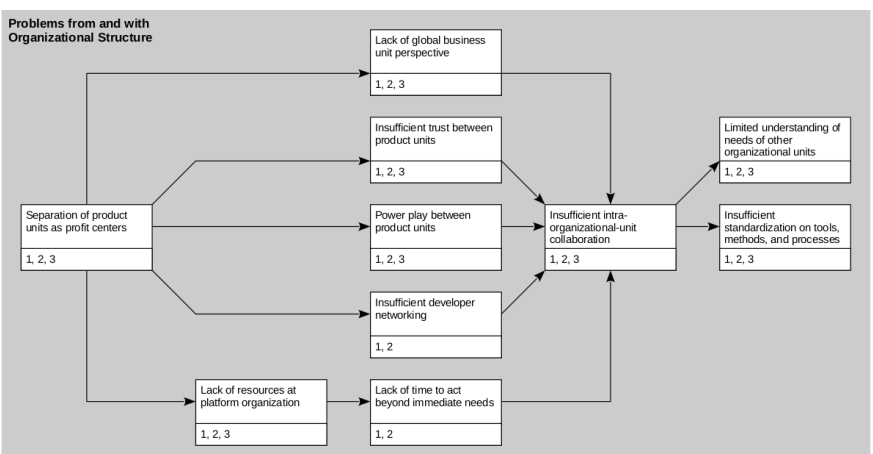
We chose these chains for presentation because they received the most mentions and interest in our interviews.

4.2.1 Chain 1: Lack of resources

In all three cases, the platform organization had a significantly higher workload than any of the product units, despite the more direct pressure on the product unit to deliver a product.

"All this work overload leads to lower code quality. I just finish up as quickly as I can and then move on." Developer (platform), case 1.





5.2.2 Hypotheses and Predictions

Our case study companies have been continuing their efforts. However, it is too early to tell whether our recommendations have been beneficial to them.

The theories we present are only as good as the hypotheses that they generate and that can be validated in future work. Such confirmatory research will also allow for generalized conclusions that are not possible from pure case study research.

H1 Resistance and misunderstandings (like expected lower code quality of inner source components) can be addressed successfully by way of education and active participation in the practice of inner source software development.

This hypothesis is likely to evaluate to true, given the change in public opinion on the use of and participation in open source soft - ware projects from a negative to a positive stance.

H2 Psychological openness or resistance to inner source (i.e. desire or fear to work under quasi-public scrutiny) depends on man - ager and developer personalities and is not a function of organizational structure or process.

...

Thus, we suggest that open source and inner source competencies are structurally similar, if not isomorphic. This is not surprising given that inner source has originally been motivated by open source. This hypothesized relationship then leads to our most potent hypothesis:

H5 While there is no doubt about the need of platform software and shared reusable assets, a platform development organization may not be needed any longer. It can be replaced by an inner source program.

This is an interesting though probably controversial hypothesis: If large companies can work together in an open source foundation to develop shared infrastructure components, why can't product units within an organization work together to create a platform of shared reusable assets without the need for a dedicated organizational unit that maintains this platform?

Supporting Evaluation, Validation, and **Use**

The Handbook Method

1. Theory creation (and later revision)
   - Creates theory using appropriate qualitative research methods, e.g. **qualitative surveys**
2. **Theory presentation**
   - **Presents theory in the form of “best practice” patterns handbooks**
3. Theory evaluation
   - Evaluates the theory using qualitative and quantitative research methods, e.g. **case study research**
4. Theory validation
   - Validates hypotheses (e.g. patterns) using hypothesis-testing methods, e.g. **controlled experiments**

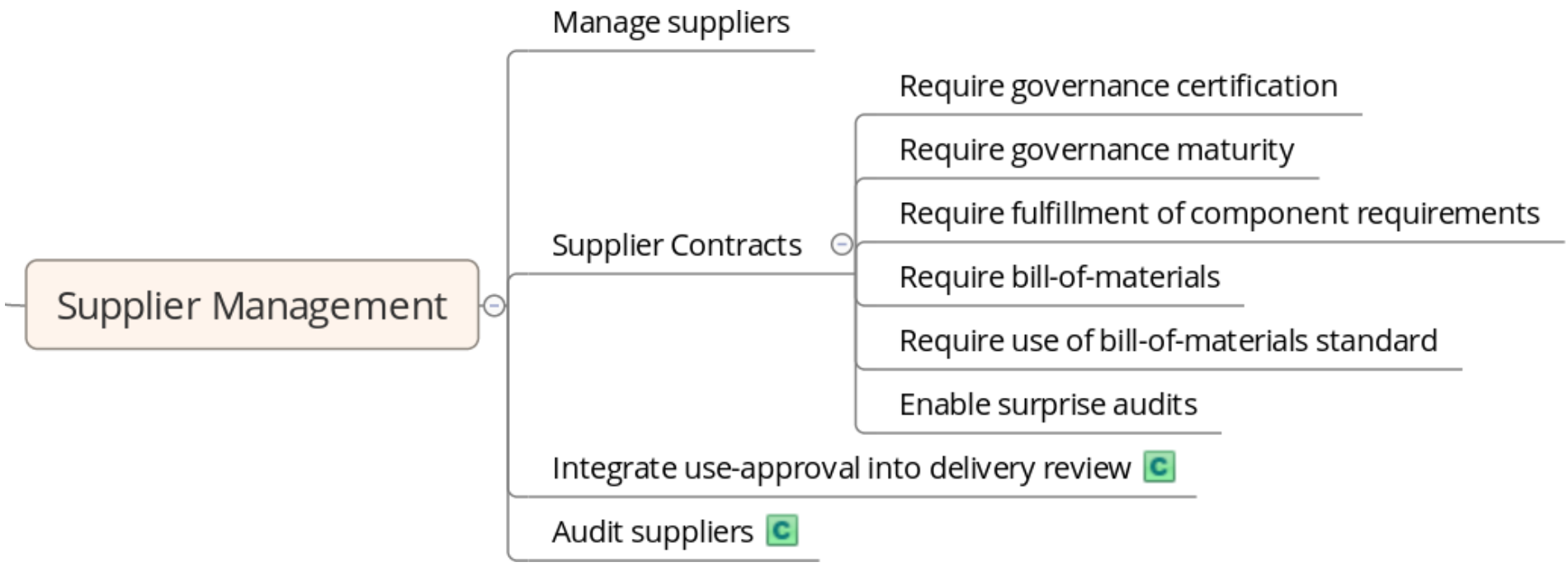
Pattern Handbooks

- Handbook
  - “A concise reference book covering a particular subject” (MW)
- (Current) best practice
  - “A procedure that has been **shown by research and experience to produce optimal results** and that is established or proposed as **a standard suitable for widespread adoption**” (MW)
- Pattern
  - “Each pattern describes a problem that occurs over and over [...], and then describes […] the solution to that problem, in such a way that you can use this solution a million times over [...]” (Alexander, 1977)
  - “Something designed or used as a model for making things” (MW)

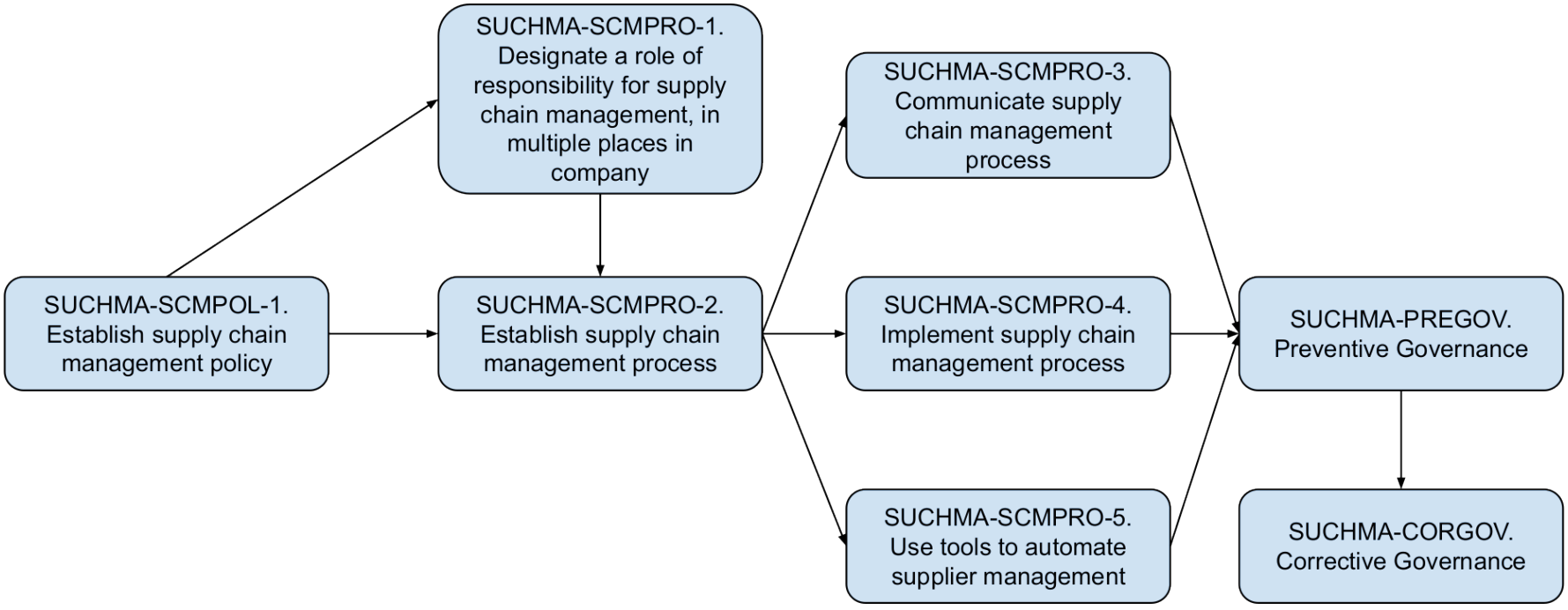
Pattern Handbook Structure

#	Content	Explanation
1	Introduction	Introduction to the handbook and overall scope
2	Roles and responsibilities	Overview of relevant roles and responsibilities
3	Domain overview	Overview of all relevant domains covered by handbook
4..n	Domain	For each domain, a detailed discussion
n+1	Conclusions	(Optional) final words

Example Domain Breakdown and Patterns



Example Workflow and Patterns [1]



[1] Examples taken from Harutyunyan, N. (2019). Corporate open source governance of software supply chains. Dissertation, Dept. Informatik, Universität Erlangen.

Pattern Format

#	Section	Explanation
1	Name	A short name for the best practice
2	Main actor(s)	Roles of people involved in the best practice
3	Context	The (abstract) context of applicability of the best practice
4	Problem	The (abstract) problem solved by the best practice
5	Solution	The actual best practice and how it solves the problem
6	Maturity	An indicator of maturity (proposed, evaluated, validated)

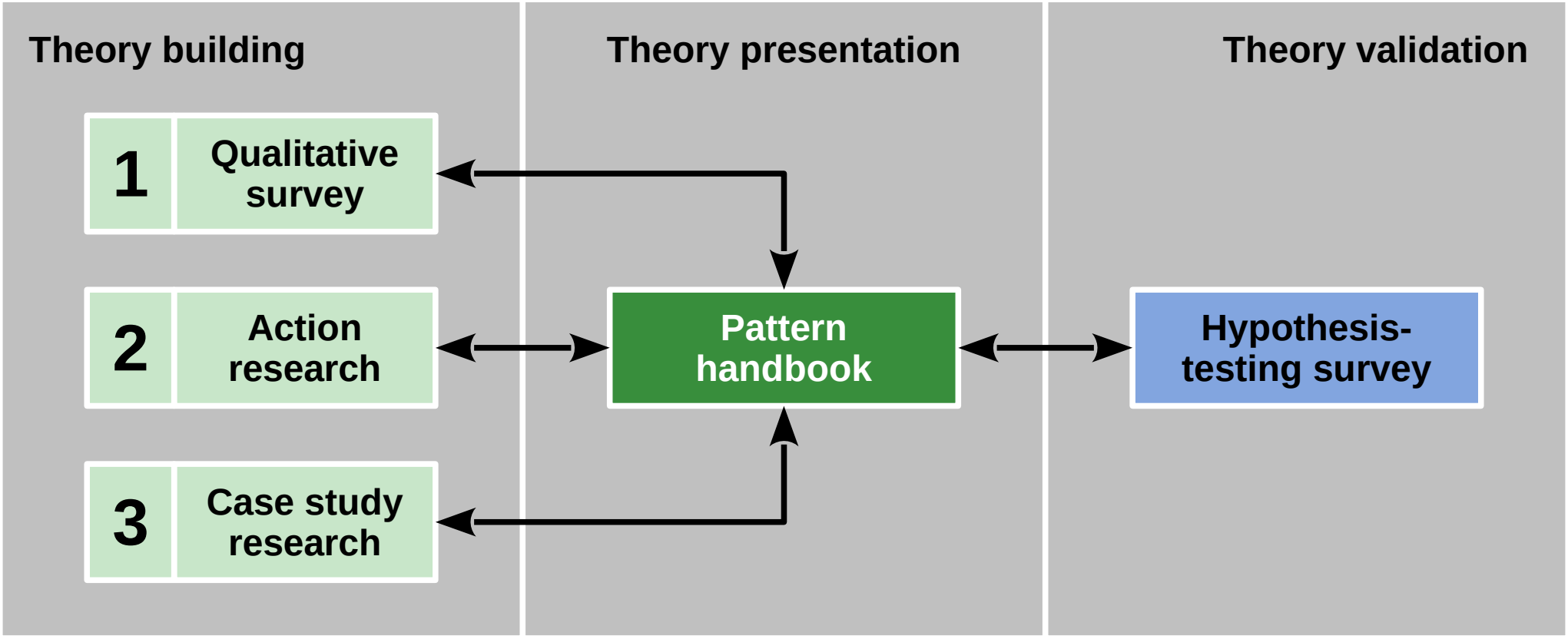
Example Pattern to “Manage Suppliers”

Name	Manage suppliers
Actor(s)	Engineering manager
Context	Your product includes not only open source components, but also third-party components that are supplied to you by other software vendors. In contrast to open source projects, you are paying for the component (license) and you are receiving it from a corporate entity. You previously → *defined (your) component requirements* and they must be met by any component, open source or not.
Problem	How to ensure that a third-party component delivery meets your requirements?
Solution	First, before you select a supplier, you may → *require governance certification* or at least → *require (a minimum) governance maturity* of them. Once you have decided for a supplier, in any delivery contract, you should → *require fulfillment of your component requirements* and you should → *require a bill-of-materials* upon delivery for which you → *require they use a bill-of-materials standard.* Upon delivery, you have to → *ensure requirements are met* and for this, you have to → *integrate use-approval into the delivery process.* If the supplier isn’t certified and having to reject a component delivery is too expensive, you may want to → *enable surprise audits* and consequently also → *perform surprise audits* as to best governance practices.
Maturity	Proposed

Scientific Community vs. Patterns Community

#	Scientific Community	Patterns Community
1	Theory creation / revision	Pattern mining / discovery
2	Hypothesis	(Proposed) pattern
3	Theory presentation	Pattern languages, handbooks, systems, ...
4	Theory evaluation	Heuristics, use of patterns
5	Theory validation	Heuristics, use of patterns

Example Research Design for the Handbook Method



Collaboration with Industry

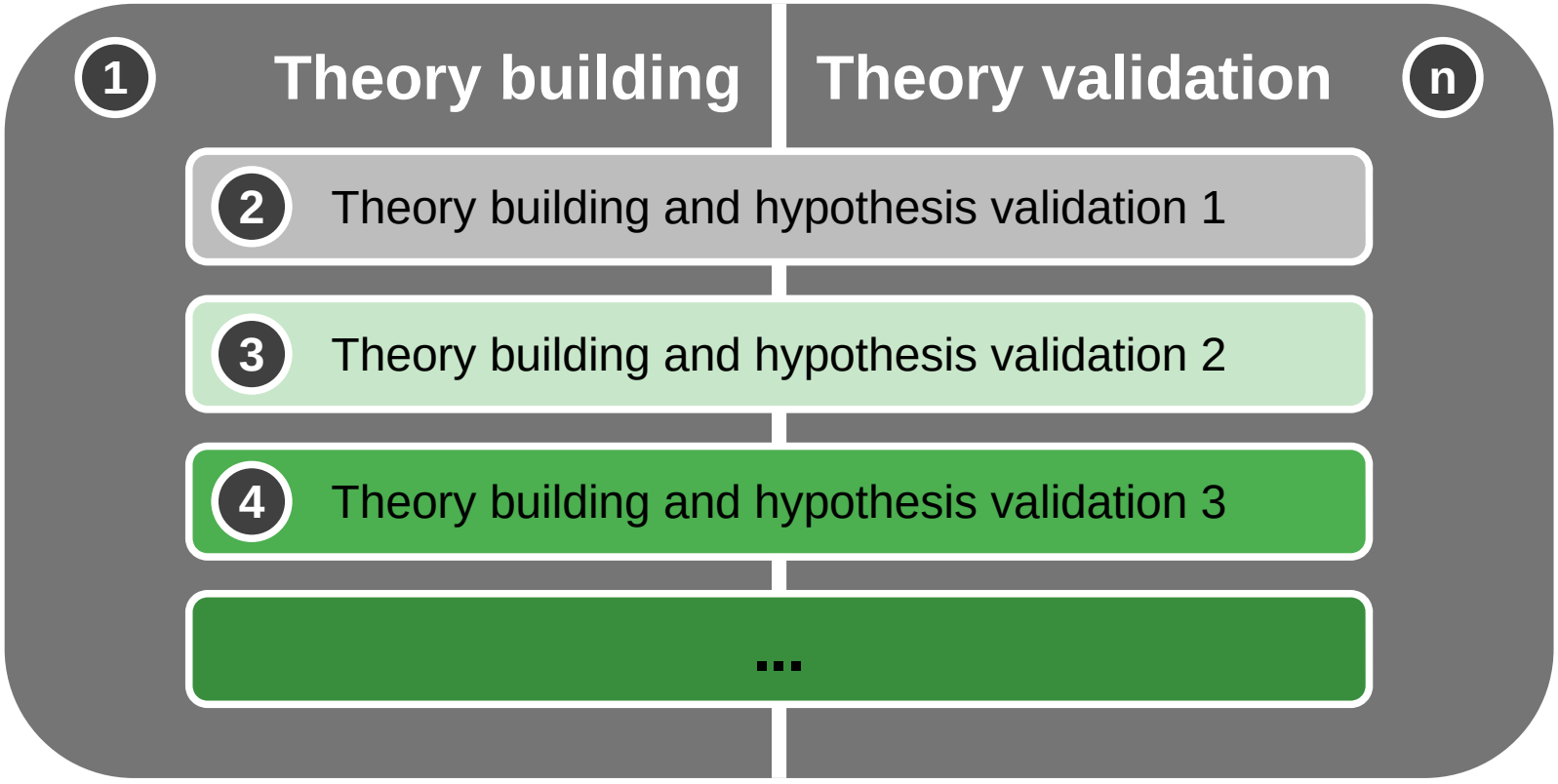
- Theory building
  - Expert interviews (qualitative surveys)
  - Researcher-guided application (action research)
  - Researcher-observing use (case study research)
- Theory validation
  - Pattern handbook testing

What About my Publications?

Publishing Patterns

- Patterns are not science…
  - Unless they have been derived using scientific methods
- Every single pattern is a hypothesis…
  - Waiting to be tested

Slicing Research Work



[1] See https://dirkriehle.com/2018/06/28/how-to-slice-your-research-work-for-publication/

More on the Handbook Method

- Riehle, D., Harutyunyan, N., & Barcomb, A. (2020). Pattern Discovery and Validation Using Scientific Research Methods. Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2020-01, February 2020.

Thank you! Questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

DR

Credits and License

- Original version
  - © 2021 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - None yet

Professorship of Open Source Software

- Professor of Computer Science
  - For software engineering and open source software
  - At the computer science department of the engineering faculty
- Previously held research positions at ...
  - SAP Labs (Silicon Valley) leading the open source research group
  - UBS (Swiss Bank, Zurich) leading the software engineering group
- Previously worked in development at ...
  - Skyva Inc. (supply chain software, Boston) as software architect
  - Bayave GmbH (on-demand business software, Berlin) as CTO

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG