# Lean Ontology Development: An Ontology Development Paradigm based on Continuous Innovation

Joel Cummings and Deborah Stacey
*School of Computer Science, University of Guelph, Guelph, Ontario, Canada*

Keywords: Ontology Development, Lean Startup, Agile Software Development, Lean Ontology Development.

Abstract: This position paper explores the utility of adapting the principles of Lean Startup and Agile software development to the development of ontologies. A main thesis is that ontology development should be approached in a manner similar to software development. Lean Ontology Development (LOD) principles are defined and current ontology development methodologies are discussed in relation to these principles. The principles defined are Continuous Development, Minimum Viable Ontology via Prioritization, Community Evaluation, Ontology as API, Reuse, and Sustainability.

## 1 INTRODUCTION

### 1.1 Motivation

While the number of ontologies is growing and the number of disciplines that have adopted the use of ontologies is also on the rise, the methodologies for the creation, maintenance and reuse of ontologies are still fragmented and largely isolated from each other. This is not to say that there are not powerful techniques that have been developed; the issue is not so much a lack of techniques but indeed there are so many tools in the toolbox that it can be difficult to discern how to structure your ontology engineering activities.

Instead of developing yet another creation methodology, this paper suggests a systems approach to ontology engineering patterned after *Agile* software development and the theory of organization development called *Lean Startup*. What is proposed is a development philosophy based on continuous, incremental improvement, competency questions, documented reuse decisions and, most importantly maintenance and sustainability planning. This approach does not restrict ontology developers from using methodologies not fully examined here (*e.g.* ontology design patterns) but suggests that they could be used in the *lean* system as long as they do not violate the Lean Ontology Development (LOD) principles described in Section 2.

### 1.2 Lean Software Development

An influential book in 2011 by Eric Ries entitled, "The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses" (Ries, 2011), introduced the idea of the lean startup. His work developed ideas from software engineering and extended it to the entire process of product development. He identified three foundations for the lean startup: (1) design thinking, (2) agile software development, and (3) lean startup method.

We will concentrate our discussion on the last two of these foundations even though the first two can have an impact on the development of ontologies. Let us start first with a definition of the lean startup method. It is a scientific approach to creating and managing startups and increasing the speed of getting a desired product into customers' hands. While Ries' original work did concentrate on the idea of startups, this approach can be and has been applied within large organizations for product development. And we are proposing that we can look at ontology development in the same way as a product for a targeted audience and that the development principles of lean startups can be an effective approach.

The lean product development process is a cycle called Build-Measure-Learn (Ries, 2018). It can be instantiated in four steps:

**Learn.** Identify the problem to be solved. As with all software (and ontology) development it is crucial that goals and expectations be articulated and do-



Cummings, J. and Stacey, D.
Lean Ontology Development: An Ontology Development Paradigm based on Continuous Innovation.
DOI: 10.5220/0006963003670374
In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 2: KEOD*, pages 367-374
ISBN: 978-989-758-330-8

cumented to facilitate both development and testing.

**Build.** Develop a minimum viable product (MVP) (Moogk, 2012). The minimum viable product is the bare basics of what you are building that is still enough to test with customers. Why *"minimum"*? One principle of lean is that the speed of development is crucial. This is not meant in a naive sense of the word. What is fast will differ between different products. Speed is related to the idea of getting feedback from users as early in the development process as possible. And while it is possible to involve users in the specifications stage with focus groups and forums, it is always more compelling to have users able to test out the product and come to a fuller understanding of how their expectations and your concepts, as embodied by the product, differ. This is a central tenet of agile software development and it is adopted by lean. The demands of producing working software (or ontology) force the development team to face issues that might not be fully explored in the specifications stage and thus will identify issues and force decisions that are best known soon than later. Thus, the true function of an MVP is to deliver your concept or ideas to get feedback from users as quickly as possible. It allows the development team to collect the maximum amount of validated learning about customers and their expectations of the product with the least effort.

**Measure.** Start measuring. Your MVP (or it subsequent variants) will allow users to work with the product. Both the development team and the users will be able to perform traditional testing, realtime monitoring, formal analysis and case studies. All of these activities will provide data and feedback to the development team in a way that is not biased by the viewpoint of the development team only. Multiple viewpoints will be brought to bear on the utility and capabilities of the product at this stage.

**Learn.** Employ investigative methods to incorporate the feedback into the product. At this stage there can be several paths available: the concepts in the current MVP can be modified to better suit the user expectations; the next version of the MVP can be developed by adding more capabilities to the product; or it can be decided that there is a need to *pivot*. As the MVP is tested and measured, it might be revealed that a hypothesis or concept about the product is wrong. If this is the case then a decision must be made to either *"persevere"* or *"pivot"* (Sekiguchi, 2018) (Eisenmann et al., 2012). This decision is critical and is often hard to accept. A pivot can be a total reimagining of the product or it could be a refocusing of the development on only a specific part of the product. In all cases, this is a difficult decision but since it will be supported by the facts in the *Measure* stage it is hard to ignore.

Lean is a principled approach to new product development. And it does not only provide a methodology to follow for the development of a product, it also encourages the development team to ask the following questions: "Should this product be built?", and "Can a sustainable business be built around this set of products and services?". It is fairly obvious that the first question can be easily repurposed to the development of ontologies. The development of an ontology is a long, hard process and thus should not be embarked on lightly. But the second question, although it looks like it is too "business-y" to apply to ontology development is actually even more important for the development of a lean ontology development methodology. It is a sad fact that although reuse is a major tenet of ontologies, many ontologies have a very short lifespan. Many are produced but few are reused. Even upper level ontologies which should be sustainable (constant evaluation, active users, new versions when needed, *etc.*) are often active for only a few years and then abandoned. So it is not outside the discussion of development methodologies to borrow from the world of commercial (and open source) software development and to treat ontologies like substantial software systems and apply a creation to deployment to retirement (*cradle to grave*) view to ontology development.

## 2 LOD PRINCIPLES

This section will briefly outline the principles of Lean Ontology Development (LOD) and will be used to develop guidelines for methodologies to instantiate these principles in practice.

One of the main tenets of LOD is that ontology development should become much more like software development. This does not imply that ontologies need to be developed by computer scientists or software engineers; the principle is that many of the established techniques used in software development are not yet fully embraced within ontology development techniques explicitly and that this can be a disadvantage.

The major design principles are:

**Continuous Development.** Since the LOD view of ontologies is that they will undergo continuous

and iterative development, the integration of versioning into the ontology development process is essential. Thus, all ontologies should have roadmaps that illustrate the versioning process so that adopters of an ontology can plan to change and grow along with the ontology. The major and minor version numbering scheme used by software should be adopted.

**Minimum Viable Ontology via Prioritization.** With the adoption of versioning it is extremely important for priorities to be established. Not all requirements or features are created equally and they should all be prioritized early in the development cycle and reviewed constantly. These priorities must be explicitly stated and documented within the development roadmap and within the ontology itself. With prioritization comes the idea of a Minimum Viable Ontology (MVO) analogous to the MVP in Lean Startup.

An example of how to realize this principle is the use of competency questions (CQs) to describe the required features of the ontology. The development of CQs is not sufficient on their own to allow for LOD. The next step to the development of CQs is their prioritization. This extra step forces the developers to think of the CQs as a whole and not as individual items in a list. Asking the ontology development team to rank or order the CQs will force discussion of what *must* be in the ontology for it to function within its imagined application domain. This will allow for the development of a *Minimum Viable Ontology (MVO)* and lead to the first iteration of the ontology that can be released to the user community. This establishes the principle of always having a *"working"* version of the ontology available to the community so that they can start evaluating it.

**Community Evaluation.** The previous principle is in line with a major principle of ontologies which is that they are a *"shared conceptualization"* (Gruber, 1995) and therefore there is a community of users. LOD indicates that this community must always be considered and involved. This community must be explicitly identified and documented in the ontology development process. Documentation can be in the form of backlogs, issues management, identification of features to be deprecated, user forums, amongst other communications mechanisms.

This evaluation by the community does not totally replace in-house testing of the ontology by the developers. It enhances the verification and validation methodologies already developed by enlarging the group of people who are contributing to this testing of the ontology.

**Ontology as API.** Ontology developers should embrace the notion of the *ontology as API*. A software API is a communications mechanism consisting of definitions, protocols and tools for constructing software. The construction of an ontology should be viewed in the same light; one cannot remove the applications of the ontology from its design and construction.

**Reuse.** It would be difficult to discuss principles of ontology design without mentioning the principle of reuse. Although considered to be a major advantage in using ontologies, in practice it is often the case that reuse is minimal or not even explicitly considered in the ontology development methodology. And there are good reasons for this - many ontologies do not lend themselves to reuse. A major factor in this is a lack of discussion on ontology latency (rate of change) and release cycles. Thus, ontology developers may feel unsure about reusing other ontologies because of the ramifications and uncertainty of changes in the reused ontology.

**Sustainability.** This principle is a culmination of all of the previous principles. The establishment of a roadmap and release cycle combined with an emphasis on the user community along with the appropriate communication mechanisms will result in a *sustainable* ontology. So many ontologies, including upper level ontologies, have not had as much impact as they could have had because they were not designed with sustainability in mind. LOD envisions a different approach that will hopefully move ontology development closer to methodologies that can achieve sustainability.

The *"take home"* message is that LOD is patterned after Agile (Highsmith and Cockburn, 2001) software development. Most current ontology development methodologies resemble the Waterfall development model. Embracing Agile will encourage the idea that ontologies are linked to communities and thus development is to be driven by the many not the few. To accomplish this goal, ontology developers must strive to always provide a *working version* of the ontology under development so that the community can experiment with it and give immediate feedback.

## 3 LOD METHODOLOGY

The following are some suggestions for existing methodologies that can be used within the LOD context. It is by no means an exhaustive list but strives to

show how various methodologies can be repurposed for LOD.

## 3.1 Development of Competency Questions

**LOD principle: Minimum Viable Ontology via Prioritization**

Competency Questions serve to specify all areas the ontology and the overall system (Fox and Grüninger, 1994) must include to be a complete product. The ontology must be able to represent the tasks needed for an external system should one be required. In developing these questions it is important to consider if and how the ontology will interoperate with other systems.

Competency questions are to be divided into basic categories as defined below but may be further subdivided into domain specific requirements to establish logical parts of the system.

- **Selection** – questions that are answered through a SELECT query on the ontologies instances
- **Counting** – questions that are answered through aggregation of instances
- **Capability** – a technology or interface the ontology must support in its design
- **Reasoning** – an aspect of reasoning that is required either through validation or categorization of instances.

In terms of further breakdown of common questions, (Ren et al., 2014) provides examples of the types of questions that are valid competency questions. It is important to remember that it must be possible for the competency questions to be answered by an ontology. As an example, questions about whether elements should be included in an ontology should be removed. Instead, concepts that should be included are derived from competency questions. In developing questions one may find redundant questions that are better answered by a more precise question. Developers should review all questions to remove redundant questions and focus on validity.

Once a set of questions is chosen, consideration of possible answers for each question should be taken. To be more precise, these answers can be thought of as return values in a function. Developers should consider cases where part of the question may become invalid due to a lack of results in an earlier part of the query; an example might be where aggregation is performed on a category without instances. Additionally answers may return both instances and values to be complete; these should be identified.

As an an example of multiple return values suppose in an ontology modelling used vehicles, one asks: "Which vehicles have the lowest price?". In this case we want both a numerical value and the instance of the cheapest vehicle. Breaking down questions like this can help in the development potential queries for selection and counting based competency questions.

Competency questions are an important vehicle for the documenting of specifications and goals for the ontology under development. A prioritization of these questions can aid in the selection of a subset of these questions to be used in the development of a minimum viable ontology (MVO).

## 3.2 Three Stage Ontology Development: Upper Level, Import, New

**LOD principles: Reuse and Sustainability**

### 3.2.1 Upper Level Ontologies

Upper level ontologies are formally defined by combining mathematical logic with philosophy to produce the most general abstraction that applies to all categories of ontology development. (Conesa et al., 2010; Kiryakov et al., 2001; Herre, 2010). Upper level ontologies define the most basic concepts, relations, and axioms to model objects within time and space. This means developers can use their expertise of the domain they are developing for without the need to become an expert in space/time. It also means one is not attempting to redefine existing agreed upon notions of time and space. Finally, there is the advantage of being able to relatively easily integrate your ontology with other ontologies built upon the same upper level ontology. This allows your ontology to interact with more than one domain or a larger part of your domain.

Upper level ontologies however, are not essential to ontology development and the first decision one must make is if an upper level ontology's qualities are applicable to your needs. Consider the following when determining whether or not to use an upper level ontology:

- Upper level ontologies require a significant investment of time to understand their design and usage.
- Ontology developers must consider the domain that they are modelling in the context of existing ontologies. Is it common to use an upper level ontology in your domain and if so what are they? Failure to take this into account can greatly impact the acceptance of the ontology by its community of users.

- If usage of upper level ontologies are not common in your domain, then check for commonality in the definition of time and space between what your ontology needs and what exists (or is duplicated) in an existing upper level ontology. If there is not enough commonality then using an upper level ontology may require you to modify these definitions thus defeating the power of using an upper level ontology.
- The ontology developer must consider their requirements for reasoning. This can help in determining if the formality and definitions for axioms of an upper level ontology may help direct this reasoning.

When choosing a particular upper level ontology, active development and usage is essential. The complexity of defining such a general design requires iterative and continuous work (Herre, 2010). This often results in the development of more than a single version. Such is the case with the Basic Formal Ontology (BFO) which is in its second major version as of this writing. Version 2 of BFO made structural changes based on community feedback and usage and the developers were responsive to user requirements which is a good sign to developers looking to adopt it.

In the case where one considers an upper level ontology to be complete and not needing changes then one should also consider new implementations and technologies. These upcoming technologies within the knowledge engineering field are often essential to ongoing works. An upper level ontology should strive to support as many technologies as possible and deprecate those that are no longer used. A sound design for an upper level ontology is one that is defined formally in first order logic with implementations in ontology specific languages. Due to differences in capability one may not meet feature parity but should strive to be as close to parity with the original design as possible. New technologies and languages often allow upper level ontologies to be implemented more precisely which can provider greater utility to prospective users. In other words, consider the development philosophy of the upper level ontology that you are evaluating.

#### 3.2.2 Importing/Using External Ontologies

The first question to ask when considering importing an external ontology is *"Why?"*. There are many reasons for doing so including: the need to use recognized vocabularies in the domain of the ontology, and to provide a mechanism for connecting to other ontologies and their communities. The exploration of other ontologies and taxonomies can serve to inspire and can be a vehicle for facilitating reuse by deciding not to *reinvent the wheel*.

But there are a number of reasons why developers are not willing to using other ontologies including the problem of totally buying in conceptually with the external ontology; there is often unease with parts of an external ontology that may dissuade adoption of even parts of that ontology. There is also the perception that it is *easier to just build it yourself* and often that is the case with the adoption of complex ontologies.

But given that reuse is considered to be a major advantage of using ontologies, how can selection of external ontologies for reuse be approached? Guidelines for safe reuse include:

- Do not just cherry pick one or two concepts or terms from another ontology. Use relations such as "owl:sameAs" and "skos:broaderThan" to reference other ontologies so that other communities can understand your ontology.
- Use *active* ontologies or *standards* like W3C ontologies to ensure sustainability of your ontology. Select ontologies that have documented versions and roadmaps and allow for community feedback.
- Connect with the highly relevant ontologies in your domain. This will facilitate community understanding and buy-in. It can often be difficult to search for ontologies so strive to identify those *portals* that contain well documented ontologies in your domain such as Bioportal, GitHub, and W3C. Identify search terms in these portals as well as in general search engines that select for ontologies in your areas of interest.
- Documentation is of extreme importance. When importing or using any external ontology, document all decisions as to why, what, and how external ontologies are used in your ontology. This will aid in sustainability and versioning.
- An often neglected part of reuse is the notion of reusing the competency questions from imported ontologies. This is often not possible because most ontologies either do not use or do not embed their CQs within themselves. If the other ontologies did not use CQs, the same effect can be had by collecting and linking to their specification documents.

These guidelines help expose the complexities of reusing ontologies and identify the need for tools to aid the ontology development analogous to makefiles and version control in software development.

#### 3.2.3 The Development of New Ontology Elements

Developing new ontology elements is ultimately the result of not finding an existing ontology that meets your requirements. This may be because an ontology exists but is large and/or includes unrelated elements from another domain that will not fit within the existing design or that a viable option does not exist. In the development of these new elements it is common for researchers to define classes and relations directly into their existing ontology. This often results in the same issue for future researchers who may be interested in a component of your ontology but are unwilling to import it due to conflict of domain content.

Instead of integrating this content directly into your ontology it is preferable to create a smaller sub-ontology that can be imported and used within your existing design. The smaller sub-ontology will define all domain specific content required for your problem. Developing in smaller sub-ontologies provides several benefits: firstly, it provides an ontology for others who may require a description of this domain in an ontology to import and use; secondly, it allows for early feedback on your design and may garner interest from experts in a community outside your own who are more familiar with the domain and will independently contribute to strengthening your initial design; and thirdly, it allows one to divide and categorize their competency questions and associate them with sub-components which can help direct users to which parts to look at when querying. Finally, the greatest advantage it provides is in the future where you and other ontology developers can use these smaller specific ontologies to develop new ontologies faster allowing one to amortize their work over time.

### 3.3 Evaluation and Sustainability

**LOD Principles: Ontology as API, Community Evaluation, and Sustainability**

#### 3.3.1 In-house Evaluation

Community evaluation does not preclude the need for in-house evaluation of the ontology by the development team. But this level of testing should be released to the community for comment and to inform community evaluation.

Evaluating the ontology requires consideration of its requirements but should also follow a high level check of competency questions, axioms and some form evaluation methodology such as FOCA (Bandeira et al., 2017). An important step to in-house invalidation is to start using the ontology internally; that means to use it with any systems or software products and generate instances as soon as possible. Ideally one has a test set of instances that can be re-used for regression testing so that each version that is released can be compared to examine breaking changes. Regression testing results can often help produce guides for others to migrate.

If the ontology is expected to be used with a reasoner, developers should be validating against the test set of instances ensuring that rules are correct and complete based on the requirements of the version.

Competency questions can be validated using example queries for selection and counting based questions. These should be validated to ensure that they return the correct instances or values and that they provide a useful response. A useful response would be a question that meets the needs of the user, *i.e.* would be a question the user would actually ask. Now is the time to reconsider the questions themselves to ensure that they make sense. Often early in the development process, requirements may not reflect actual usage requirements due to missing information or misunderstandings of user needs.

#### 3.3.2 Releases and Versioning

Developing a standard for releasing and versioning the ontology should be established early in development. The standard should include documentation on release number schemes, release note contents, and methods for user contribution. In scheduling releases one should consider the competency questions particularly in order since one can define a preliminary road map of where they fit.

Versioning should use some form of major and minor versioning which denoted using a standard naming scheme. Minor versions should include small additions, bug fixes or other errors within the ontology. It should also include some form of annotations that denote elements that will be deprecated or changed within the next major version. These annotations will allow users to work with the ontology and avoid using soon to change terminology and also contribute to replacement terms. In other terms, minor versions should not make breaking changes to the interface as in software libraries. In ontological terms this means users should be able to use the same classes and relations they were using in the prior minor version.

Major versions may include breaking changes that alter the structure, remove terms, and or include sets of competency questions to meet some milestone. Within each major or minor version any competency questions implemented and/or terms added or deprecated should be included in the release notes as well. Major versions should only include significant feature

inclusions or alterations to the ontology otherwise minor versions should be used.

Prior to starting the next version one may choose to publish a tentative road map that lists competency questions that will be included within the next release. Community feedback may alter the order or priority of issues.

Finally one must remember that their ontology likely depends on other ontologies which may make breaking changes to their design. This requires tracking versions of required source ontologies in the way that package managers and build scripts do in software. A similar approach will need to be taken for source ontologies to track versions used and breaking changes. Throughout time these ontologies will need review to ensure newer versions are still compatible with usage. External ontology changes then make an impact on future releases as they will influence things like deprecated terms, *etc.*.

The release cycle consists of the following:

- Develop - Evaluate/Testing - Repeat (Agile methodology)
  - evaluating and testing by community, *e.g.* similar to the way BFO does this
- Versioning by adding more CQs
- Versioning of dependent ontologies - need the concept of a *makefile* for ontologies. This is currently done by documentation alone.

# 4 CASE STUDY

A brief example of how this approach can help is illustrated by our experience as part of a large team developing a complex ontology on culture with the Canadian Writing Research Collaboratory (Brown, 2018). A very large and detailed list of competency questions was compiled for the ontology (although not before extensive preliminary ontology development was done). This has served the ontology development process well but it also exposed to us a danger in not documenting all requirements explicitly. One of the tools being used to visualize the ontology has limitations on what relations it can handle at the current time (it is also under development) and so if the ontology cannot be visualized by the tool this might limit its use by the user community. This necessitates that this requirement to conform to this tool should be in the CQs. Because this requirement is not in the CQs this means that we did not prioritize this and thus decisions based on the usability of the tool are addressed outside of the CQs and could result in incomplete documentation of design decisions. The CQs are meant to provide documentation for design decisions and so must include every aspect of the use of the ontology including tools that will be used with the ontology. This tool identification aspect is particularly important since the development of quality tools for ontologies is in its infancy and the decision to link design decisions to tools must be considered carefully. This does not mean that a decision on using a particular tool is wrong, in fact, it might be essential for community acceptance. But documenting this decision via CQs is essential for the development of an MVO and acceptance by the user community. Identification of this tool requirement and its documentaton has aided us in our MVO development.

Another benefit was the discussion generated by this issue and the realization that although the use of the visualization tool was desirable it was not to be used as an argument against specific ontological elements if they possessed other distinct advantages. A mature approach to prioritization has been developed via this discussion.

The need for evaluation, both in-house and by the user community, has resulted in the development of a testbed of linked data that can be used to evaluate the utility of the ontology to describe the domain area. This testbed is being developed in-house but has involved external consultations with external researchers. The development team has also started to survey (personal consultations and on-line survey) the user community with regards to their familiarity with and usage of existing ontologies. This will help the team to make choices with regards to the use of existing ontologies. One example of this is the exploration and decision making around the representation of *events* in the ontology. The team has examined many existing *Event* ontologies including CIDOC-CRM's Event class, W3C's Event ontology and DOLCE Lite's event handling. Discussions with the user community is ongoing to establish which Event ontology will be used by the CWRC ontology. The Agile/Lean principle of consulting and engaging the user community throughout the entire development process has definitely helped to inform the ontology development work and to excite the user community with the possibilities of this new Digital Humanities ontology.

# 5 CONCLUSIONS

This short discussion paper on adapting Lean Startup and Agile Software Development to ontology development aims to inspire discussion about the intersection of ontology development and software engi-

neering. There are many areas of similarity between the two domains and it appears that ontology development and the tools to enable it are at a point where they can benefit from the appraoches and tools involved in software development. The principles of Continuous Development, Minimum Viable Ontology via Prioritization, Community Evaluation, Ontology as API, Reuse, and Sustainabilty are a start to the development of a Lean Ontology Development paradigm that takes lessons and tools from software engineering and adapts them to the needs of ontology developers.

## ACKNOWLEDGEMENTS

This work has been greatly improved by the authors' discussions and collaboration with Shawn Hind and the following members and staff of CWRC: Susan Brown, Kim Martin, Abi Lemak, Jasmine Drudge-Wilson, Alliyya Mohammed, and Gurjap Singh.

## REFERENCES

Bandeira, J., Bittencourt, I., Espinheira, P., and Isotani, S. (2017). Foca: A methodology for ontology evaluation. https://arxiv.org/abs/1612.03353v2. Accessed: 2018-06-18.

Brown, S. (2018). Cwrc/csec — canadian writing research collaboratory. https://beta.cwrc.ca/. Accessed: 2018-06-18.

Conesa, J., Storey, V. C., and Sugumaran, V. (2010). Usability of upper level ontologies: The case of researchcyc. *Data & Knowledge Engineering*, 69(4):343 – 356.

Eisenmann, T. R., Ries, E., and Dillard, S. (2012). Hypothesis-driven entrepreneurship: The lean startup. *Harvard Business School Entrepreneurial Management Case No. 812-095*.

Fox, M. S. and Grüninger, M. (1994). Ontologies for enterprise integration. In *CoopIS*.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, 43(5-6):907–928.

Herre, H. (2010). General formal ontology (gfo): A foundational ontology for conceptual modelling. In Roberto Poli, Michael Healy, A. K., editor, *Theory and Applications of Ontology: Computer Applications*, pages 297–345.

Highsmith, J. and Cockburn, A. (2001). Agile software development: the business of innovation. *Computer*, 34(9):120–127.

Kiryakov, A., Simov, K., and Dimitrov, M. (2001). Ontomap: portal for upper-level ontologies. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 47–58.

Moogk, D. R. (2012). Minimum viable product and the importance of experimentation in technology startups. *Technology Innovation Management Review*, 2(3):23–26.

Ren, Y., Parvizi, A., Mellish, C., Pan, J. Z., van Deemter, K., and Stevens, R. (2014). Towards competency question-driven ontology authoring. In Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., and Tordai, A., editors, *The Semantic Web: Trends and Challenges*, pages 752–767, Cham. Springer International Publishing.

Ries, E. (2011). *The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses*. Portfolio, London.

Ries, E. (2018). The lean startup methodology. http://theleanstartup.com/principles. Accessed: 2018-06-18.

Sekiguchi, Y. (2018). Lean startup - what is pivot? http://www.slideshare.net/YukiSekiguchi/lean-startup-what-is-pivot. Accessed: 2018-06-18.