Agile Scalability Requirements

Gunnar Brataas and Tor Erlend Fægri
SINTEF Digital
Trondheim, Norway

ABSTRACT

Many software organisations struggle to provide appropriate levels of scalability in their software systems. Agile development rests on pragmatic and value-centred approaches to requirement capture that allows customers and vendors to interact in the process of producing the software system that best mets the real needs of the customers. In collaboration with Norwegian software organisations we have observed that setting scalability requirements is hard. Organisations struggle because they lack a conceptually sound language for expressing scalability requirements. To improve current practice, we propose a light-weight and flexible approach to specifying scalability requirements. Flexibility ensures that a more extensive characterisation can be used if higher precision is required, and more information becomes available.

CCS Concepts

•**Software and its engineering** → **Agile software development;** *Requirements analysis;* •**General and reference** → *Performance;*

Keywords

Agile development; Requirements engineering; Metrics

1. INTRODUCTION

In this paper we refer to *scalability* as the ability of a service to increase its capacity by consuming more hardware and software resources [5]. Software resources are lower-level software services like DBMSs, and with hardware, we mean CPUs, disks and networks. These resources all have a cost, which may be expressed as dollars per hour.

Capacity is the maximum workload a service can handle as bound by its SLA (Service Level Agreement) [5]. Together, work and load constitute the workload of a service. A scalable software service can satisfy more customers (load) or more demanding customers (work) by consuming more software and hardware resources. In their interaction with software-based services, people have little patience – regardless of the number of users of a service or service complexity. Software vendors must provide appropriate scalability despite rapid innovation cycles or requirement volatility.

We have established a research project named ScrumScale [1] who's objective is to reduce the total cost of handling scalability using a tailored, agile development approach. ScrumScale involves the collaboration with three Norwegian software organisations: 1) EVRY is a leading Nordic supplier of software-based solutions and services within the financial and other sectors. 2) Powel is a provider of software solutions for the energy, public and contracting sectors 3) Altinn is the largest Norwegian public portal, for example used by the tax authority to deliver digitised public services.

ScrumScale is now in the initial phase, and our work has so far focused on understanding how these organisations address the scalability challenge when developing software-based services. All three organisations have extensive scalability knowledge, but nonetheless, suffer from expensive re-engineering because of unaddressed scalability requirements. Like many other organisations, they often end up with significant accumulation of technical debt with the costly consequence that scalability must be retrofitted at the end of the project. Why does this happen? We have identified vague and unclear scalability requirements as one potential root cause.

Agile development is fuelled by rapid feedback loops and accepts simplicity in written documents as a trade-off for more 'live' understanding of the system under development. Agile development encourage collaboration among stakeholders to bridge understanding from both the problem-space and the solution-space. To support a more collaborative approach to scalability engineering, we need a light-weight, yet reasonably accurate approach to capture scalability requirements.

In this paper, we describe a conceptual model for scalability. The model constitutes a language for expressing scalability requirements. We deliberately keep the number of scalability parameters low. We also describe granularity trade-offs so that the scalability requirements can accommodate different contexts yet remain conceptually consistent.

In Section 2, we will summarise some early findings from our three Norwegian software organisations. Our initial approach for describing scalability requirements is outlined in Section 3. In this section, we also describe important granularity trade-offs. State of the art is described in Section 4. Conclusions and further work are offered in Section 5.

ICPE'17 April 22-26, 2017, L'Aquila, Italy

ACM ISBN 978-1-4503-4404-3/17/04.

DOI: http://dx.doi.org/10.1145/3030207.3030240

2. EARLY FINDINGS

We have collected data at all three organisations using semi-structured interviews, observation of regular performance review meetings, project retrospectives, available documentation and meetings with participants from the organisations. The following is a brief summary of the findings.

We have learned that software organisations often choose to emphasise functional requirements that provide visible and tangible value to their customers. Non-functional requirements, such as scalability, receive less attention. The ownership of scalability requirements is frequently unclear. Limited collaboration when defining scalability requirements leads to vague scalability requirements, e.g. "our solutions shall scale vertically and horizontally." Vagueness in scalability requirements imply that they rarely need revision, and they therefore remain outside the center of attention.

Because scalability requirements are vague and ownership is unclear, testers must sometimes guess and make their own proposal for testable scalability requirements. Architects and developers may then be working on new projects and subsequently become less accessible. Valuable insights from testers and operations personnel are also easily lost when designing new services. We speculate that misplaced and diffuse ownership of scalability requirements is a fundamental explanation why scalability requirements are subjected to such ad-hoc treatment.

However, we observed an interesting paradox. There is a large gap between theory and practice. Several of the people we interviewed commented that the scalability concepts we propose are valuable and even obvious. On the other hand, these concepts are not used in practice. Project after project fails to deliver sufficient levels of scalability.

We conclude that managing scalability is difficult in our three case organisations. We believe that the complexity is probably not buried in the individual concepts, but in the lack of a systematic approach to apply *the set of concepts* relevant for scalability. To achieve this, we need a lightweight and flexible approach to define scalability requirements based on a conceptually sound model of scalability. Moreover, actors with clear ownership of scalability requirements need to be involved early in development.

3. SCALABILITY REQUIREMENTS

Our method for capturing scalability requirements focuses on the concepts shown in Figure 1. A service delivers one or more operations, each offering a unique way of interacting with the service. Stakeholders are responsible for formulating critical operations with quality metrics and quality thresholds. They have to decide on projected maximum load for these critical operations, where operation mixes play an important part, as they describe the probability of each of the operations. Work parameters describe the amount of data used by the critical operations. All these information is part of scalability requirements.

According to our scalability definition in Section: 1, (theoretical) scalability is about the ability to increase capacity. This ability is vital. However, if the cost of achieving a certain capacity is too high, this may severely limit the *practical* scalability. We will first consider the important theoretical scalability requirement concepts. Afterwards, we will also introduce cost so that practical scalability can be explored.

Scalability focuses on steady-state behaviour. In practice,

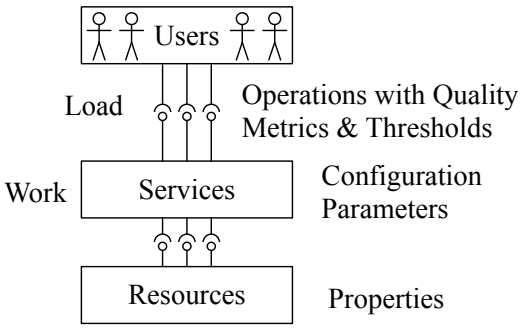

Figure 1: Scalability concepts.

especially open load varies heavily. Cost efficiency is about the total cost also when fluctuations in work and load are taken into account, but is outside of our scope in this paper.

3.1 Maximum Values in Planning Horizon

According to our scalability definition, we should investigate how our service responds to increasing workloads as well as stricter SLAs. To simplify this task, we focus on capturing the toughest values for work, load and SLAs, during a given planning horizon. A planning horizon of less than three years will seldom be meaningful when developing new software. A simple scalability analysis uses the maximum values for work, load and SLAs. A more complex scalability analysis may also consider even tougher requirements.

3.2 Quality Metrics and Thresholds

A quality metric defines how we measure a certain quality and is a key part of an SLA (Service Level Agreement). Generally, all the operations will share the same quality metrics for each operation, e.g., average response times or 90 percentile response times. However, with many operations, they we may of course also use several quality metrics.

Quality thresholds (QTs) describe the border between acceptable and non-acceptable quality for a given metric, e.g., 2 seconds. For each quality metric for each operation, there must be one quality threshold.

3.3 Critical Operations

In the ideal case, we should carefully consider the scalability implications of *all* operations for a service. However, there may be several hundred relevant operations. We must therefore simplify, and focus on the critical operations. The critical operations are the operation that from a scalability point of view poses the largest risks for SLA violations.

This poses a difficult dilemma, however. Critical operations are those operations that are most influential in determining the scalability of the service. Unfortunately, their influence can only be determined accurately after the service is developed and is deployed in operation. As a result, we cannot know with confidence which operations will be critical, but must select critical operations from careful reasoning. Confidence will increase by expanding the number of critical operations, but at the expense of the human effort involved in formulating the scalability requirements.

We must at least implicitly know the quality metrics and thresholds to determine the critical operations. With a long enough response time, an operation will never be critical.

3.4 Load and Operation Mix

Load describes how often the operations in a service are invoked. A closed system has a constant number of users. Load for a closed system is specified by think time (Z) and simultaneous number of users (N). An open system has a variable number of users and is characterised by the arrival rate λ. Based on the number of simultaneous users, we can derive the number of concurrent users actually using the same lower-level software service at the same time. Today closed systems are generally more challenging compared to online systems, simply because a system only becomes a batch system if it is too heavy for being an online system.

For simplicity, let us first assume that we have an open system with only one operation. Load is now characterised by the arrival rate for this operation. We are typically interested in the maximum load, but often more investigations are required to find this maximum load. To think about how the arrival rate fluctuates according to seasonal and trend variations may be helpful; not because of all the details, but to find the maximum load.

Seasonal variations can be yearly, monthly, weekly or daily. During one typical day, load may be highest immediately after people arrive at work at 9AM. In a week, load may be highest on Wednesdays. For a month, load may reach its maximum near the end of the month. During one year, load may be highest just before Christmas. A trend may be a linear or exponential growth with, for example, 10 % every year. Assuming a three year planning horizon, we can expect the highest load to be just after 9AM on a Monday around December 20 in the third year.

With more operations, load can be specified for each operation individually, or we may specify the load on the average operation. An operation mix describes the probability of each of the operations. We define the operation mix for the critical operations to sum to 1. As a result, the probability of one operation depends on which operations are included in the set of critical operations.

If some of the critical operations refer to an open system, and others to a closed system, we get a group of critical open operations and a group of critical closed operations. Both these groups have a corresponding operation mix. For the open operation mix, when a new user arrives, the operation mix represents the probability of the different operations. A heavy operation will last longer than a light operation. Therefore, the number of simultaneous operations will not reflect the operation mix. In a closed system, when one operation is completed, the closed operation mix represents the probability of the new operations.

3.5 Work Parameters

Work characterises the amount of data to be processed, stored, or communicated by the operations in a service, e.g., the average size of a document or the number of documents. A service may have anything from none to several work parameters.

We are interested in the maximum values for each work parameter, e.g. the highest number of documents or the highest average size of documents. For work parameters, we have tree assumptions: 1) Seasonal variations are not relevant. 2) Trends are relevant. 3) Work parameters grow so that we get the highest work values towards the end of the planning horizon.

3.6 Practical Scalability

Ideally, we should not focus on implementation when describing requirements. However, if we are interested in cost, we have to consider lower-level software resources as well as amount of and cost of hardware. We also have to consider the mapping between our service and the lower-level software and hardware resources. Will this mapping be linear, and if not, which form will it take? Moreover, scalability will also be determined by the configuration parameters for the lower-level software services and hardware, but this is very detailed information, and may therefore be ignored.

3.7 Feedback Loops

For hard challenges such as scalability, it is often necessary to cycle extensively between working in the problem space and the solution space. Collaborative practice involving stakeholders from both domains can support this, for example, guided by techniques such as proof of concepts, prototyping and incremental demoes. Agile development methods embrace such cycling. For example, to formulate quality thresholds, the critical operations must be known. To determine the critical operations the quality thresholds are required. Hence, this information must be developed in close collaboration driven by feedback between the parties. Information from operations from similar services can help.

3.8 Granularity

By granularity, we mean level of detail. It is common practice in agile development to ensure mutual understanding of requirements in dialogue between the product owner and the development team taking responsibility of implementation. In this dialogue, pending implementation, we expect that the level of detail in scalability requirements can be a core topic. As we have mentioned before, there are inherent cyclic dependencies between the problem- and the solution-space in complex development tasks. The scalability requirements will evolve together with the understanding of both the solution- and the problem-space. Our scalability requirements will therefore themselves be agile, as well as support an agile development method.

Our suggested prioritised list of dimensions of granularity can be a useful conceptual origin for such dialogue:

- **Decomposed operations:** One operation may be decomposed into several lower-level operations. As a result, load and quality thresholds must also be detailed.
- **Lower level services:** We may also have scalability requirements for lower-level services, which may be formulated in the same way as the service scalability requirements. For example, a certain operation from a lower-level service has a quality threshold with a quality metric guaranteed for a maximum load and for a given value of work parameters.
- **More work parameters:** Initially, only a few work parameters may be relevant, but gradually more can be introduced, e.g. we may start with number of documents (one work parameter) and afterwards add average as well as maximum document size (two more work parameters).
- **Richer quality metrics:** The quality metric may be implicit, e.g. an organisation will always assume

the average response times metric, unless otherwise is clearly stated. By explicitly detailing the quality metrics, the level of granularity as well as the accuracy of our requirements will increase.

- **Detailed quality thresholds:** Instead of the same quality thresholds for all operations, we can specify individual quality thresholds for each operation. There may also be more quality thresholds for each operation, e.g. good, fair, bad or contract violation (with corresponding penalty fees).

We also assume that there will be context-specific demands to the level of precision that the scalability requirement will be specified to. Secondly, there will be variations in the amount of information that is available. For development projects using agile methods, flexibility in precision can allow requirements with sufficient levels of precision for the specific context. The following list of aspects can be adjusted according to specific needs:

- **Can we cope with one critical operation?** Quality threshold, implicit quality metric (e.g. average response times) and maximum load required. No work parameters specified.
- **Can we cope with several critical operations as well as critical work?** Above plus: Individual quality thresholds for all critical operations, operation mix and work parameters.
- **What is the resulting cost of the critical operations?** Above plus: mapping from workload to amount of lower-level services (a coarse model or based on measuring a running system — both options may require considerable manual effort in practise). Hardware costs (bare metal or IaaS) and costs of lower level software resources (licences or costs for PaaS/SaaS). We may be satisfied if the cost grows approximately linear with the load and work, assuming the cost of the initial configuration is acceptable.

4. STATE OF THE ART

Among our Norwegian partners, use of reactive performance tools is widespread. The tools are predominantly supporting application load testing and performance monitoring. Prolonged stress testing is also used to identify memory leaks, etc. that potentially could create performance problems during use. When testing scalability, load generation tools (JMeter, LoadRunner etc.) are heavily used. Load generators depend on precise specifications of work, load, quality metrics and quality thresholds. Profiling tools are used, but less frequently. Our partners do not make models of software and hardware.

Still, there are fundamental problems with this reactive approach. Scalability is a quality that transcends the concerns of the individual user. It is an aggregate quality that can best be observed at high workloads when many users compete for resources.

There has been significant research interest in the ability of agile methods to accommodate 'technical requirements' in a balanced manner to avoid the accumulation of technical debt [3]. However, to our knowledge, scalability has not yet been addressed specifically by researchers in agile methods. Nevertheless, relevant insight from this field suggests that agile practises are able to accommodate non-functional requirements as long as certain considerations are taken seriously [2]. Nord et al. [6] suggested the merging of both functional and architectural requirements incrementally during development. This was coined 'the zipper model' to agile architecting.

Duboc et al. [4] describe a general framework for scalability requirements, while we focus only on scalability related to software and hardware resources. We make our scalability assumptions *explicit* by describing them as operations, load profile, work, load and quality thresholds. In contrast to Duboc, we make the granularity trade-off explicit and focus on the usefulness of our approach in a practical, industrial setting.

5. CONCLUSIONS AND FURTHER WORK

In this paper, we have outlined a conceptual framework for capturing scalability requirements. Going forward, we seek to address the following research questions: (1) Is the presented conceptual model of scalability requirements useful for software organisations in practice? (2) What processes are necessary to effectively support the generation of scalability requirements? (3) To which extent are our scalability requirements testable? (4) How do we manage the evolution of scalability requirements for long-lived systems?

Much work remains to be done. Agile developers are accustomed to capture requirements in user stories. We need worked out examples where we apply our concepts and describe the challenges met and how we deal with them, for example, by detailing our scalability requirement concepts. If our conceptual model of scalability requirements is effective, tailored tools should be developed further to streamline an agile development process.

6. ACKNOWLEDGMENTS

The research leading to these results has received funding from the Norwegian Research Council under grant #256669 (ScrumScale).

7. REFERENCES

[1] ScrumScale. www.scrumscale.com. Visited: 20 February 2017.

[2] M. A. Babar. *Agile Software Architecture*. Morgan Kaufmann, 2014.

[3] W. N. Behutiye, P. Rodriguez, M. Oivo, and A. Tosun. Analyzing the Concept of Technical Debt in the Context of Agile Software Development: A Systematic Literature Review. *Information and Software Technology*, 82:139–158, 2017.

[4] L. Duboc, E. Letier, and D. Rosenblum. Systematic Elaboration of Scalability Requirements through Goal-Obstacle Analysis. *Transactions on Software Engineering*, 39(1):119 – 140, 2013.

[5] S. Lehrig, H. Eikerling, and S. Becker. Scalability, Elasticity and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics. In *Conf. on Quality of Software Architectures*. ACM, 2015.

[6] R. L. Nord, I. Ozkaya, and P. Kruchten. Agile in Distress: Architecture to the Rescue. In *XP Workshops*. Springer, 2014.