vision & approach to software engineering

Lodewijk Bergmans

Summary

Software engineering is about continuously balancing three key dimensions: *technology*, *people*, and *value*:

- **technology**: how to apply SE techniques to manage software development.
- **people**: how to empower people to work effectively and manage complexity.
- **value**: the continuous economical trade-offs that must be considered; are the costs worth the benefits, are development efforts delivering value to the customer.

Our unique approach is to 'calibrate' this balance using the vision and techniques of **scalable design**, with a toolbox of common, state-of-the-art and unique composition techniques, the latter developed in recent years at the University of Twente.

A scalable design has the ability to enhance the design model of a system by adding or modifying functionality or features with minimal effort (impact). In other words, it is a *design* that is *scalable* with respect to evolving requirements, growing features and functionality, changes and extensions, and alternative implementations for different environments and platforms.

Table of Contents

Economics of software development: 3

Scalable Design 4

Why do we need scalable design? 4

Vision on Software Engineering 5

I. Technology 5

II. People 5

III. Value 5

Impact of scalable design on software engineering 6

I. *Technology* toolbox for scalable design 6

II. Achieve scalable design with *people* 8

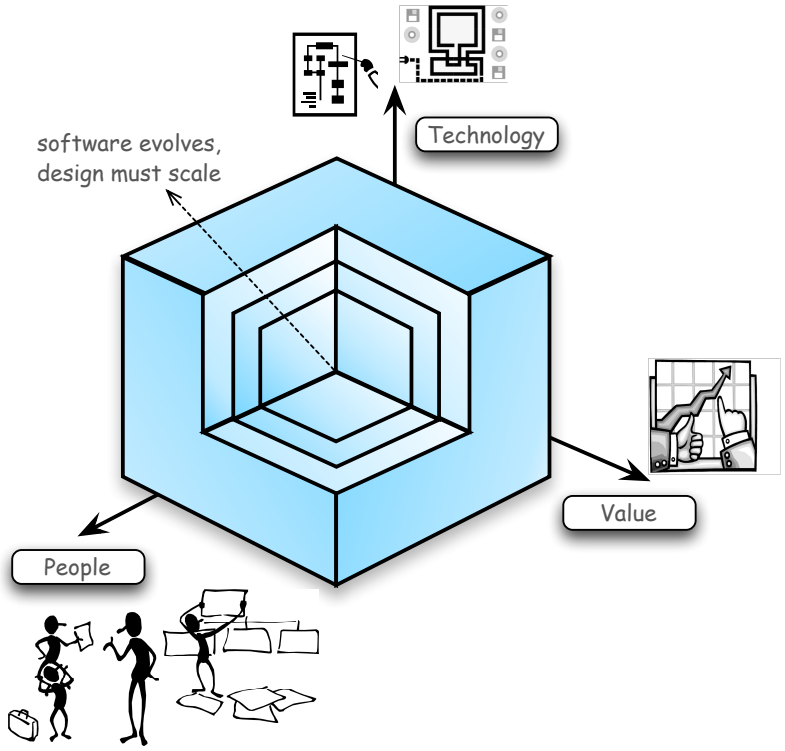
III. Use scalable design to deliver *value* 9

Design a scalable software development organization 10

Pragmatic software engineering consulting 11

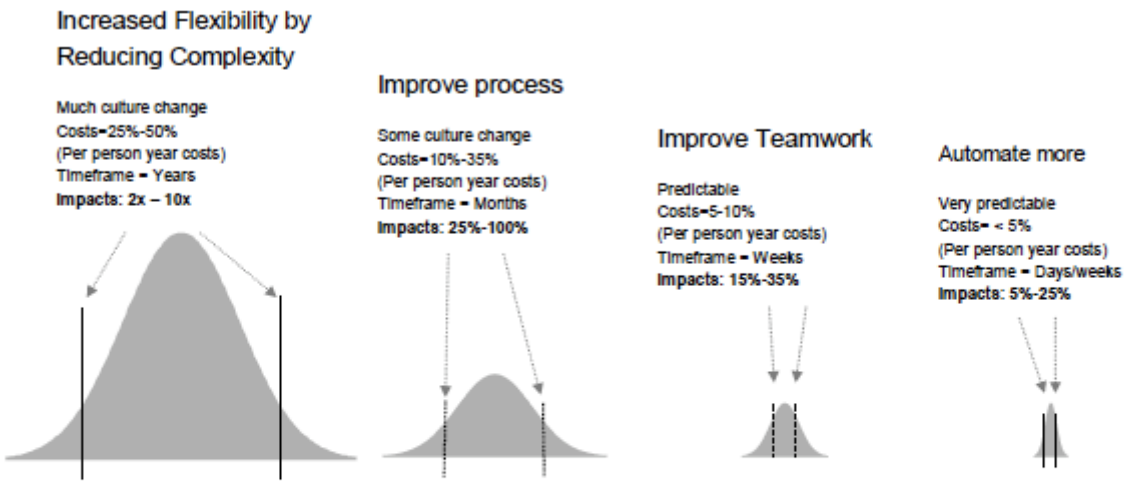
Approach 11

Consulting services 12



Economics of software development:

The following picture is derived from a 2009 IBM white paper by Walker Royce, titled “Improving Software Economics”:



The purpose of this figure is to illustrate the potential impact of 4 types of improvement in software development activities; each of these require an investment, and successfully addressing that issue has a prognosed impact; the message of the white paper is that the best way of approaching software improvement is by a balanced attack on all dimensions. However, we would still like to point out the huge potential of complexity reduction.

Scalable Design

A scalable design refers to the ability to enhance the design of a system by adding or modifying functionality or features with minimal effort (impact).

Note that we emphasize the design itself; the structure and conceptual model of a system, rather than how the performance of a system can be maintained as its load increases (i.e. scalable performance). However; a good scalable design should *also* scale with respect to different performance requirements and their implementations!

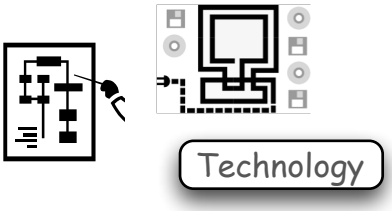
Why do we need scalable design?

- **continuous change**: software designs are not cast in concrete, and have never been. Now, more then ever (and even more so in the future), the world changes, so the requirements of software change, evolve and grow: on average, 70% of the development effort takes place after the initial delivery. This requires a software design that scales with the stream of change requests and features.
- **managing complexity**: The study by IBM we mentioned on page 3 proposes that of all possible productivity improvements, the potential gains of reducing complexity are up to 10 times higher than other improvements (and in fact estimated to yield up to 10x productivity improvements). Managing complexity requires that the software design is structured such that new or changing requirements can be incorporated without significantly affecting the structure of the system.

Vision on Software Engineering

Software engineering is sometimes considered a technical discipline. However, this is only partially true; truly effective software development projects require careful attention to, and balancing of human factors, and economical considerations as well. We summarize these three disciplines with the keywords *technology*, *people* and *value*:

I. Technology



Software development is clearly a technology intensive discipline; both inside our products, and during product development, we apply a lot of technology, which is rapidly changing all time. The technology dimension includes the models, languages and design techniques that are used in software development.

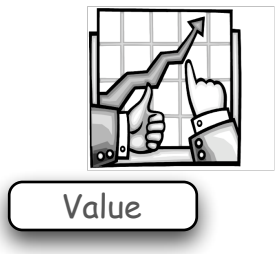
Software engineering requires us to consider how to use technology–with care–to build better and cheaper products, and how technology can support software development activities.

II. People

The people dimension addresses how people learn, create, build models, manage complexity, make mistakes, and work together while constructing software systems.

This dimension considers what are suitable processes, methods, notations, tools and languages to support people in the software development activities.

III. Value



Value, costs and economics are unavoidable factors to consider in any professional activity. Software engineering must consider the costs and benefits involved in developing a product; this applies to both the products, and the process, by which the product is constructed, maintained and evolved.

Costs are the eternal trade-off factor that we must take into account; it impacts the choice of technology, processes and product features. In the end, we need to consider the expected benefits in terms of (long-term) profits and the associated costs.

Impact of scalable design on software engineering

The realization of scalable design impacts many aspects in software development. Here is how it affects the three dimensions of software engineering we presented before:

I. *Technology* toolbox for scalable design

In the technology dimension, the software engineer needs a large toolbox of methods and techniques; we have unique knowledge, skills and experience in this area. The following are common techniques for achieving scalable designs:

- **(architecture) design methods**: this is a key element; only a careful design of the architecture can achieve a scalable design. Important characteristics are identification of stable key concepts, separation of concern and composability of architectural abstractions.
- **product lines**: a software product line is a set of software systems that share common features and that are developed systematically from a common set of core assets. It is a key enabler for achieving better reuse, shorter time-to-market and better reliability with reduced efforts.
- **model-driven engineering (MDE) and domain-specific languages (DSLs)**: these are modern approaches that enable software development at a higher level of abstraction; closer to the problem domain, while reducing the amount of (repetitive) code and improving platform independence.
- **frameworks**: (application) frameworks are well-organized software libraries that capture the essential and common abstractions in a specific domain, allowing for substantial reuse while retaining full flexibility.

(continues on next page…)

(continuation of common techniques for scalable design:)

- **modular software**: *the* key enabler for a scalable software design is the ability to define and compose individual building blocks; to this end, many techniques exist, several of these can be used concurrently:
  - **component infrastructure**: computing platform interfaces that package typical units of deployment such as .NET assemblies, Java Beans and OSGI bundles, CORBA and COM components.
  - **plug-in models**: allow for 'graceful promotion' of a system by incremental extensions; for example Eclipse plug-in architecture, OSGI bundles. This is especially interesting for third-party extensions in a software platform/ecosystem context.
  - **composition techniques**: we have extensive knowledge of and experience with many forms of **object-oriented** and **aspect-oriented** composition.
  - **Domain Specific Composition technology**: a new paradigm in constructing highly flexible *and* modular systems; this is particularly interesting if your software is offering its users capabilities for aggregating or combining parts. Ask us about it!

II. Achieve scalable design with *people*

Achieving scalable design may involve substantial changes to organization, process and the skills and mindset of both software engineers, project managers and stake-holders (!). The pragmatic way to achieve this is by adopting a *scalable improvement process*:

- Create a 'scalable design mindset': continuous awareness that every activity, and every next deadline is just a small step on the grand scale of a long-lived system, and if done well, its results may last for a long time, and in multiple contexts.
- Start by improving the tools and way of working, such that these match the available skills and organizational context.
- adopt many of the best practices from agile software development, such as test-driven approaches, continuous refactoring and short incremental development cycles.
- an appropriate software development infrastructure is an important enabler: version management, build environment, tool support for automated testing, etcetera.
- Scale up the knowledge and skills of involved people through teaching, mentoring, leadership, so that they are mentally equipped to create scalable designs.

III. Use scalable design to deliver *value*

The goal of scalable design is to ensure that created software artifacts keep being (re-) used, in other words, to maximize the value per effort. Along the same lines, it is important to prioritize the artifacts that are of the most value to the customer.

- Hence, scale the developed product to the circumstances; this combines very well with the scalable design practice of centering the design around the core concepts of the solution.
- Incremental (lean) development: first build the features that deliver the most value. Note that incremental development can only scale up if the underlying architecture is sufficiently stable and flexible!
- Enable and exploit reuse of software artifacts to avoid unnecessary efforts:
- by creating reusable abstractions
- by adopting a product line approach; this enables systematic reuse between similar products, even while these products keep evolving!
- through the adoption of other techniques, such as application frameworks, DSLs, as discussed in the section 'I. *Technology* toolbox for scalable design' on page 6.

Design a scalable software development organization

In a certain sense, one has to *design* a feasible development process by taking into account, and balancing, the three dimensions of technology, people and value. As in any design activity, this involves making continuous trade-offs; what are the expected benefits vs. costs of a long-lived system? what is the (total) cost of adopting advanced tools and techniques, and how much benefit do they offer? Do we need, and expect, all persons to rise to an expert level? How many simultaneous innovations can we manage within a single project?

We aim to co-operate with its customers to identify the trade-offs, and weigh the involved concerns to make deliberate decisions on how to improve the software engineering activities of the customer.

Pragmatic software engineering consulting

Approach

Our view on pragmatic software engineering consulting is to carefully consider all three dimensions when deciding how to improve a project or an organization. Then, address these dimensions in a sequence of steps;

- **listen**: first, understand the problems, domain and forces in the customer context. Second: keep on listening as time continues..
- **understand**: analyze the origin of the problems: are they related to process or methods, way of working, adopted tools and techniques, or the design decisions that have (not) been made? How do they relate to the goals of the business?
- **solve**: propose solutions that offer a careful weighing of the factors technology, people, and value.
- **consolidate**: make sure that the proposed solutions are consolidated; for example as architectural and design documentation, as documented process guidelines, by a tool adoption process, or by a teaching and mentoring trajectory.

Pragmatic software engineering is about continuously considering the three dimensions, not focusing on only technological, process or economical improvements and ignoring the others.

Consulting services

We offer the following services:

- **design & architecture consulting**: contribute to, or lead, software (architecture) design activities
- **design assessments (audits)**: identify risks, provide recommendations
- **improve way of working**: develop a way of working that is tailored to the skills and practices of the customer, and introduce it into the organization, for example with the help of:
- **mentoring & courses**: consolidate knowledge and skills in an organization by tailored courses and on-the-job mentoring.
- **composition technology consulting**: tap our expert knowledge on e.g. advanced object-oriented and aspect-oriented composition techniques, and our experience in introducing these in organizations.

We are happy to discuss risk sharing and fixed price alternatives!