Module Product Families and Generic Developments

Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

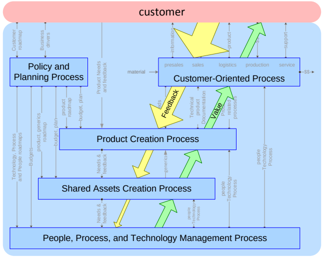
This module addresses product families and generic developments.

version: 1.3 status: preliminary draft

Contents

Chapter 1

Product Families and Generic Aspects


customer
Customer roadmap
Business drivers
Product Needs and feedback
information
product
$$
support
Policy and Planning Process
material
presales
sales
logistics
production
service
$$
Customer-Oriented Process
Technology, Process and People roadmaps
Budgets
product, generics roadmap
budget, plan
product roadmap
budget, plan
Feedback
Value
Technical product Documentation
Product related processes
people Technology Process
Product Creation Process
Needs & feedback
generics
Shared Assets Creation Process
Needs & feedback
people Technology Process
People, Process, and Technology Management Process

1.1 Introduction

Platform

Common components

Standard design

Framework

Family architecture

Generic aspects, functions, or features

Reuse

Products (in project environment)

Figure 1.1: Different names for development strategies that strive to harvest synergy

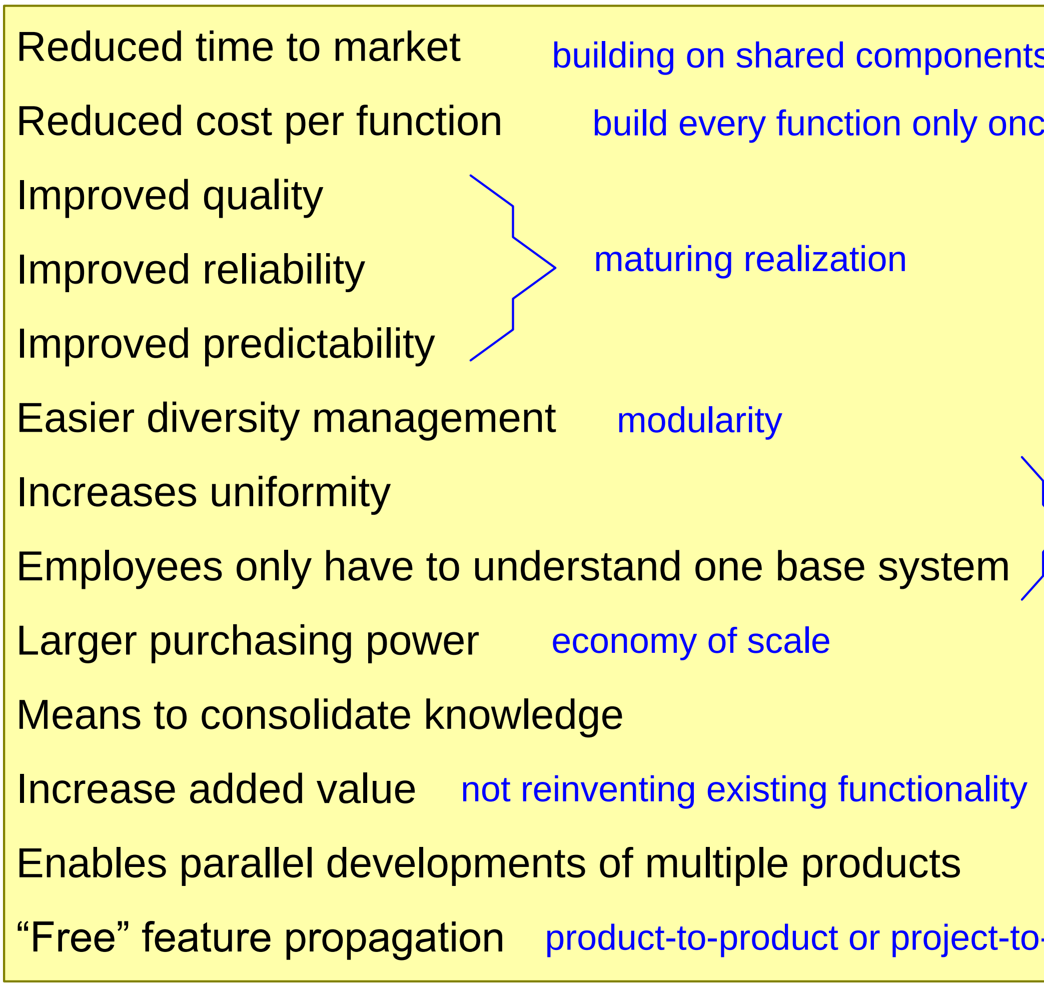
Harvesting synergy between products or projects is being done under many different names, such as shown in Figure 1.1. We us as label for this phenomena. The reader may substitute the name that is used in their organization.

Many trends (increased variability, increased number of features, increased interoperability and connectivity, decreased ti globalization of markets) in the world force organizations into these strategies where synergy is harvested. Harvesting synergy organizational and technical. We strive to give insight in both needs and complications of harvesting synergy, in the hope that establish an effective synergy harvesting strategy.

1.2 Why generic developments?

Many people advocate generic developments, claiming a wide range of advantages, such as listed in Figure 1.2.

Effective implementation of generic development has proven to be quite difficult. Many attempts to achieve these claims b opposite of these claims and goals, such as increased time to market, quality and reliability problems et cetera. We need a better r



Reduced time to market building on shared components

Reduced cost per function build every function only onc

Improved quality

Improved reliability

Improved predictability

maturing realization

Easier diversity management modularity

Increases uniformity

Employees only have to understand one base system

Larger purchasing power economy of scale

Means to consolidate knowledge

Increase added value not reinventing existing functionality

Enables parallel developments of multiple products

“Free” feature propagation product-to-product or project-to-

Figure 1.2: Advantages which are often claimed for generic developm

to design an effective Shared Assets Creation Process.

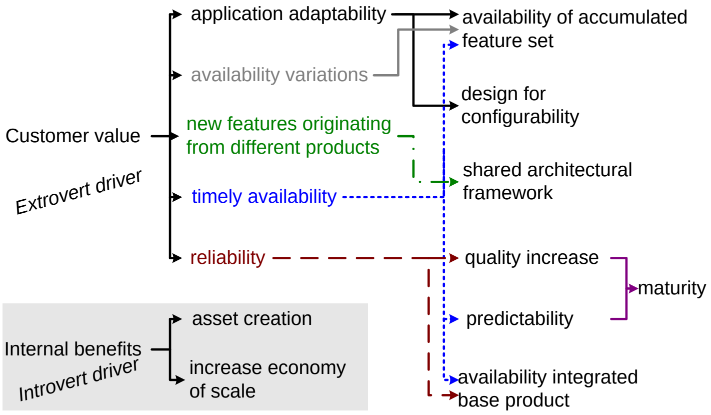

Customer value
Extrovert driver
application adaptability
availability variations
new features originating from different products
timely availability
reliability
availability of accumulated feature set
design for configurability
shared architectural framework
quality increase
predictability
maturity
availability integrated base product
Internal benefits
Introvert driver
asset creation
increase economy of scale

Figure 1.3: Drivers of Generic Developments

Figure 1.3 shows drivers for Generic Developments and the derived requirements for the Shared Assets Creation Proc
the product have value for the customer and is the customer willing to buy the product? The second driver *Internal Bene*
a company.

Today high tech companies are know how and skill constrained, in a market that is extremely fast changing and r
constraint that has to be balanced with the capability to create valuable and sellable products.

The derivation of the requirements for the product development shows that these requirements are not a goal in itsel
instance, a shared architecture framework is required to enable features developed for one product to be used in other pr
it creates value for a customer. So the verification of the shared architecture framework requirement has to involve the p
using limited effort and lead time.

We emphasize the derivation from drivers to requirements because many generic developments fulfil the requirements
for configurability, *shared architectural framework*, and *maturity or implementation*, without bringing the assumed

developments result in large monolithic solutions, without flexibility and long development times. Developers of such framework
have this easy shortcut, because our architectural framework does not support it, changing the framework will cost us 100 man-yea

1.3 Granularity Of Generic Developments

Granularity is one of the key design choices for systems architects: what is an appropriate decomposition level for modularity?
levels for different purposes. For example, in the application granularity of functions and roles, at specification level granularity
granularity of functions and concepts, and in implementation granularity of many operations.

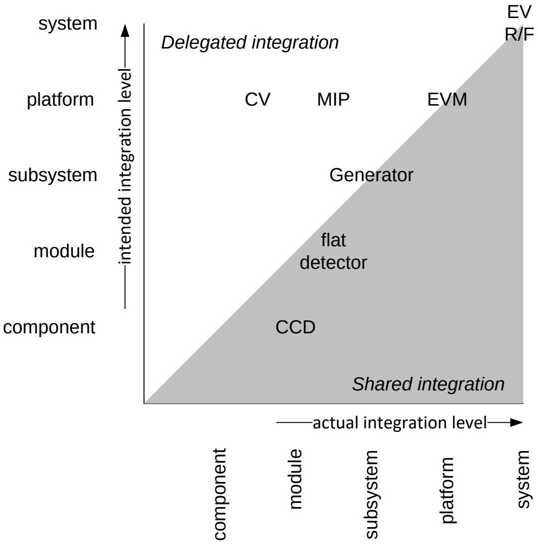


Figure 1.4: Granularity of generic developments shown in 2 dimensions.

Figure 1.4 shows the granularity of generic developments in 2 dimensions. The vertical dimension is the preparation level
developments, how far is the deployment prepared? The horizontal dimension is the integration level: How far are the generi
developers deploy the generic development?

Both axis range from (atomic) component until (configurable) system. Developments on the diagonal axis, which have a s
the integration level, are straightforward developments in which the integration takes place as far as autonomously possible. So
generation of building blocks, leaving ("delegating") the integration to the product developer. For rather critical generic develop
goes beyond its own deliverable to ensure the correct performance of the asset in its future context(s).

In these figures a number of medical generic developments are shown, as an example for the categorization.

An extreme example of "delegated" integration is Common Viewing (CV). The organization made an attempt to harvest syne
to create a large "toolbox" with building blocks that could be used in a wide variety of medical products ranging from Magnetic
systems. A powerful set of (mostly SW) components was created, using Object Oriented technology and supporting a high degree

The CV toolbox proved difficult to sell to product developers, amongst others due to the low integration level. The perception of
to do the majority of difficult work: the integration. The vision of a marketing manager changed the direction of CV into creating
Radiography Fluoroscopy (EV RF). This medical workstation for the URF (Universal Radiography Fluoroscopy) market was hig
server. The communication and print function were highly configurable to make the product adaptable to its environment.

The EasyVision RF was used as a basis for a whole series of medical workstations and servers. The shared functionality is d
level. This platform is nowadays called EasyVision Modules (EVM). Despite its name it has still a significant integration level
bothered with the lower level integration) and its downside (predefined functionality and behavior).

The old CV vision is revived and a second generation of EVM is being created, covering the EVM platform functionality with fi
The whole evolution as described here from CV as toolbox to more fine grained EVM modules took about 15 years. During al
(degree of sharing) and customer value has been changing without ever achieving the combination of a high degree of sharing and

1.4 Modified Process Decomposition

In **??** we discussed a simplified process description of companies. This decomposition assumes that product creation processes for m
When generic developments are factored out for strategic reasons then an additional process is added: the Shared Assets Creation P
modified process decomposition

Figure 1.6 shows these processes from the financial point of view. From financial point of view the purpose of this additiona
These assets are used by the Product Creation Process to ensure the cash flow for the near future by staying competitive.

The consequence of this additional process is an lengthening of the value chain and consequently a longer feedback chain as we
length of the feedback chain is a significant threat for generic developments. The distance between designers and developers of sha

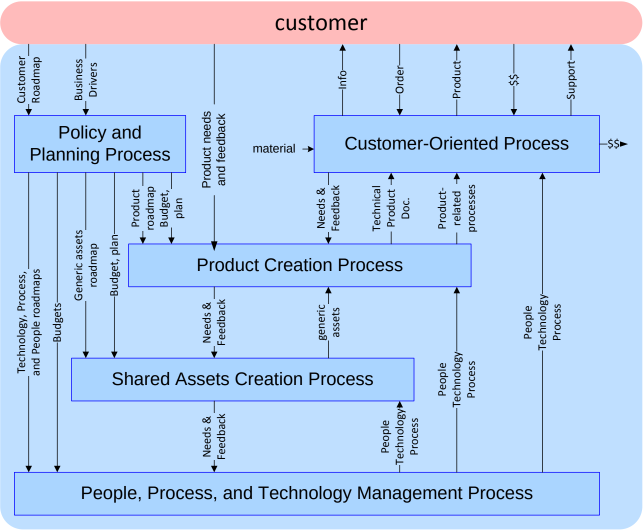

customer
Customer Roadmap
Business Drivers
Policy and Planning Process
Product needs and feedback
Info
Order
Product
$$
Support
material
Customer-Oriented Process
$$
Product roadmap
Budget, plan
Needs & Feedback
Technical Product Doc.
Product-related processes
Generic assets roadmap
Budget, plan
Product Creation Process
Technology, Process, and People roadmaps
Budgets
Needs & Feedback
generic assets
People Technology Process
People Technology Process
Shared Assets Creation Process
Needs & Feedback
People Technology Process
People, Process, and Technology Management Process

Figure 1.5: Modified process decomposition

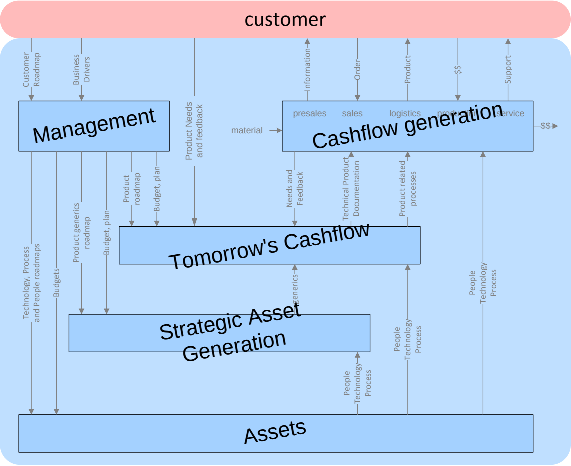


Figure 1.6: Financial viewpoint of processes

world is large. These developers easily lose focus on customer value and may focus on the technology instead. Successf
value and technology.

1.5 Modified Operational Organization of Product Creation

The operational organization of the Product Creation Process is described in **??**. This organization is a straightforward hie
between products or subsystems are managed at the closest hierarchical management level.

Introduction of generic developments complicates the operational structure significantly[1]. Figure 1.8 shows the ope
with the necessary additions to support generic developments.

The conventional Product Creation Process is based on a relative straightforward hierarchy, where the control flow a
the hierarchy. The introduction of generic developments breaks this simple structure: a generic development team delive
is taking place from an encompassing operational level, to enable operational balancing of products and generic developm
is not the customer anymore, but an intermediate manager.

Every operational entity needs the 3 complementing processes in the product creation process: operational managem
processes a role is required of someone responsible for that process: the operational manager, the architect and the comm
team of the operation. Introduction of generic developments also requires the introduction of these roles for the shared as

For the architect role this means that a platform architect is needed, who is closely working together with the plat
other hand the platform architect needs many architectural contacts with the product family architect, acting as the arch
customers, and with the component architects, acting as suppliers.

The separation of the roles of the platform architect and the product family architect is not obvious. For example in [1
are identified. Application Family Engineering (AFE), Component System Engineering (CSE), and Application System E
Component, and Product as shown in Figure 1.8. We will either have a gap or a double role, when mapping 4 operational

[1] The complication can be avoided by working sequentially. However in today's dynamic market sequential work results in unacceptable lead
looking for opportunities to reduce the lead time more.

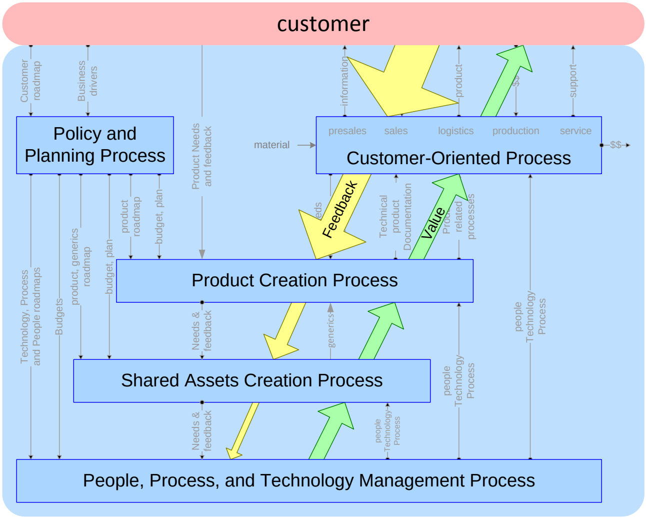

customer
Customer roadmap
Business drivers
Policy and Planning Process
Product Needs and feedback
material
information
product
$$
support
presales
sales
logistics
production
service
Customer-Oriented Process
$$
eds
Feedback
Technical product Documentation
Value
Product related processes
product roadmap
budget, plan
product, generics roadmap
budget, plan
Technology, Process and People roadmaps
Budgets
Product Creation Process
Needs & feedback
generics
people Technology Process
people Technology Process
Shared Assets Creation Process
Needs & feedback
people Technology Process
People, Process, and Technology Management Process

Figure 1.7: Feedback and Value flow

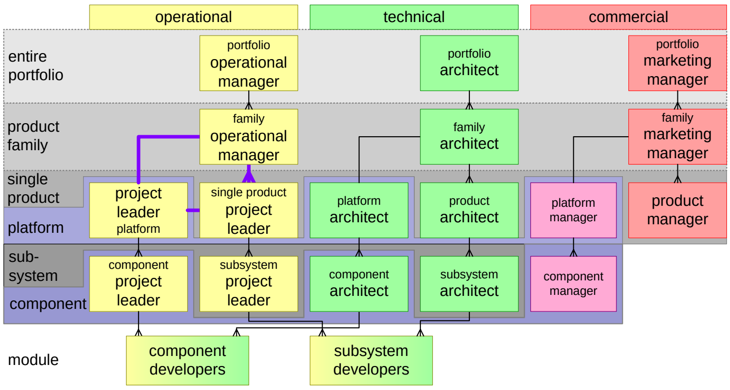


Figure 1.8: Operational Organization of the Product Creation Process, modified to enable generic

of the roles is missing, or played implicit. For instance quite often the application family engineer starts to play platform architect, engineering. We have observed that architects either tend to play the platform architect role or the product family role. Architects

1.6 Models for Generic Developments

Many different models for the development of shared assets are in use. An important differentiating characteristic is the drivin organization structure. The main flavors of driving forces are shown in figure 1.9.

1.6.1 Lead Customer

The lead customer as driving force guarantees a direct feedback path from an actual customer. Due to the importance of feedback disadvantages of this approach are that the outcome of such a development often needs a lot of work to make it reusable as a gene and performance parameters of the lead customer, while all other functions and performance parameters are secondary in the b customer can be rather customer specific, with a low value for other customers.

1.6.2 Carrier Product

The combination of a generic development with one of the product developments also shortens the feedback cycle, although the feed Combination with a normal product development will result in a better coverage of performance parameters and functionality. D takes full ownership for the product (which is good!), while giving the generic development second priority, which from family po

In larger product families the different charters of the product teams create a political tension. Especially in immature or pov counterproductive political games.

Lead customer driven product development, where the product is at the same time the carrier for the platform combines the product approach. In our experience this is the most effective approach of generic developments. A prerequisite for success is an political games.

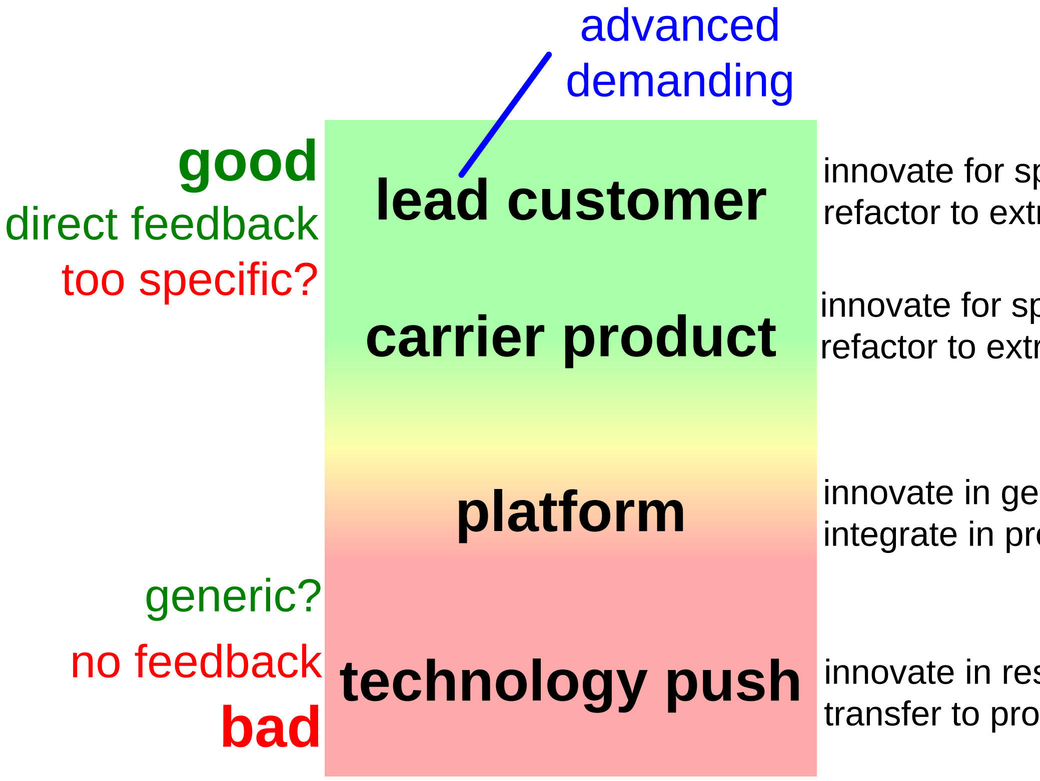


Figure 1.9: Models for SW reuse

1.6.3 Platform

Generic developments are often decoupled from the product developments in maturing product families, by creating an a where integration plays a major role (nearly all products) the shared assets are pre-integrated into a platform or base p release process before it can be used by product developments.

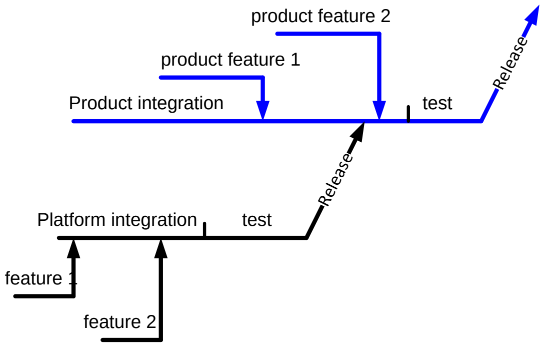


Figure 1.10: The introduction of a new feature as part of a platform causes an additional latency i

The benefit of this approach is separation of concerns and decoupling of products and platforms in smaller manage of such a model: as a consequence the feedback loop is stretched to a dangerous duration. At the same time the dura figure 1.10.

1.6.4 Alternative Generic Development Scenarios

A number alternative re-use strategies have been applied with more or less success:

Spin-out as an independent company is especially tried for key and base technologies. However, many spin-out comp Examples are multimedia processors from TriMedia (parent Philips Semiconductors, later NXP) and cell phone op

Reuse after use works quite good in practice, especially for good clean designs.

Opportunistic copy where implementations are taken that are available. The results are quite mixed. Short term bene Longer term a problem can be that an architectural mess has been growing that turns into a legacy.

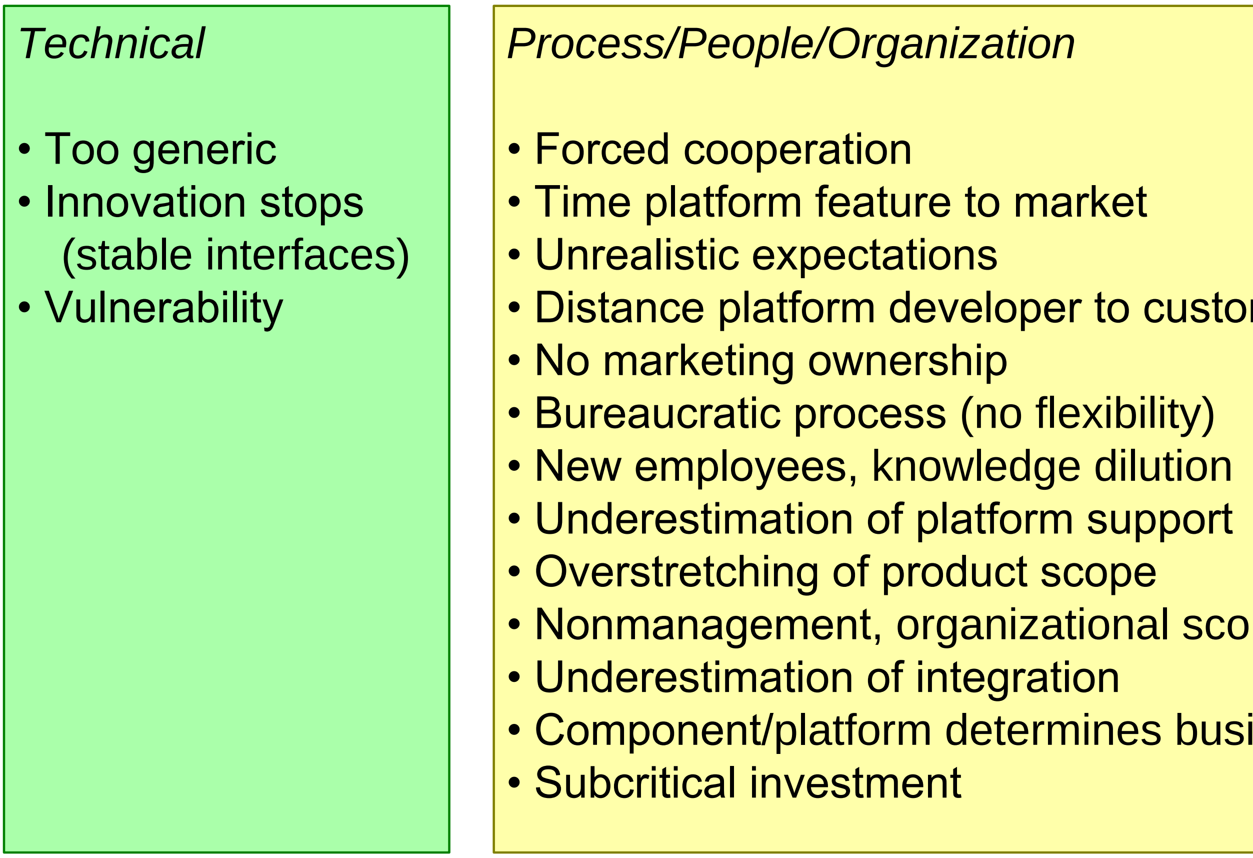
Open source where key and base technologies are shared and developed much more publicly.

Inner-source , where a company stimulates sharing takes place within a company modeled after an open source approach.

volutionary refactoring where the architecture and its components are actively re-factored to keep them fit for the future and for

1.7 Common Pitfalls

We learn from out mistakes. Unfortunately, many mistakes have been made in the area of generic developments. We compiled mistakes in generic developments in the past. Some of the attempts to harvest synergy were partially successful, but issues from th



Technical

- Too generic
- Innovation stops (stable interfaces)
- Vulnerability

Process/People/Organization

- Forced cooperation
- Time platform feature to market
- Unrealistic expectations
- Distance platform developer to custo
- No marketing ownership
- Bureaucratic process (no flexibility)
- New employees, knowledge dilution
- Underestimation of platform support
- Overstretching of product scope
- Nonmanagement, organizational sco
- Underestimation of integration
- Component/platform determines busi
- Subcritical investment

Figure 1.11: Sources of failure in generic developments

Most of the problems have a root cause in people, process, or organizational issues. The list with technical problems is relative

Too generic platform or components that can do everything, but nothing really good: “the Swiss army knife”

Innovation stops , because existing interfaces are declared to be stable. Existing structure and interfaces can block innovation.

Vulnerability , because all products use one and the same core. If the shared core has a problem anywhere then all products are h that enhances resilience. In nature, species often survive disasters, such as diseases, due to the diversity in the population.

Forced cooperation by upper management, de-motivating employees, and creating social and political tensions in the organizatio

Time platform feature to market because of stacked release procedures.

Unrealistic expectations by upper management, often as a consequence of the claims from architects and engineers o the benefits than promised, then a negative spiral sets in of cost reduction and hence even more decreasing outcome.

Distance platform developer to customer , see Figure 1.7.

No marketing ownership , but engineering push only. Marketing support is crucial, since marketing is one of the key players wh of marketing ownership results in a continuous fight for funding, with starvation in the end.

Bureaucratic process , and loss of flexibility. The increased scope of the operation (common components or platform plus de organization than the individual products used to have. The formalization easily turns into bureaucratism, slowing down the

Knowledge dilution caused by the hiring of new employees. Often an increase in resources is needed early during th inexperienced, then the knowledge is diluted, resulting in less quality of the created assets.

Underestimation of shared asset support required when the shared assets are used by products. Product designers nee based on these assets, and they need support for trouble shooting during integration and introduction in the field. new products), then always unexpected problems pop-up.

Overstretching of product scope beyond the natural level of synergy. Harvesting synergy is a balancing act, betwee minimizing diversity in the realization. When the minimization of diversity dominates over value creation, the business. Organizations easily lose their customer focus, when creating a synergy drive.

Non-management of organizational scope increase that is inherent when multiple products share assets. The scop adaptations.

Underestimation of integration of shared assets in other products. Systems integration is often ill understood and henc to migrate to the use of shared assets, then this requires that these products adapt their architecture too.

Component/platform determines business policy which is effectively an inversion of the need driven approach. This development and customers. What happens is that what *can* be done dominates over what *needs* to be done. Th products depend on their delivery.

Subcritical investment , caused by a cost reduction focus. Shared asset development primarily should bring market harvesting synergy. As soon as cost reduction dominates over value creation, then all products and shared assets c problems.

1.8 Acknowledgments

During the first CTT course system architecture, from november 22 until november 26 1999, a lively discussion about g input for this article. I am grateful to the following people, who attended this course: Dieter Hammer, Wil Hoogenstraa Maurice Penners, Pierre America, Peter Jaspers, Joost Versteijlen, Peter Beelen, Jarl Blijd, Marcel Dijkema, Werner Bandakka, Jodie Ledeboer

I thank Pierre America for working on consistency in spelling and the use of capitols. Ad van den Langenberg pointe

Bibliography

[1] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse; Architecture, Process and Organization for Business Succes*

[2] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

History

Version: 1.3, date: March 9, 2015 changed by: Gerrit Muller

- repaired copy/paste remainder

Version: 1.2, date: October 19, 2014 changed by: Gerrit Muller

- added summary

Version: 1.1, date: July 6, 2004 changed by: Gerrit Muller

- removed Product Families presentation and article

Version: 1.0, date: March 25, 2004 changed by: Gerrit Muller

- created reader