Chapter 11. Conclusions

This section contains the conclusions of this thesis and is structured as follows: First the answers to the research questions are given, which is followed by a discussion on the threats to validity. Then hypotheses are presented on the applicability of this research to other domains. This section ends with the key contributions of this thesis and a description of further research.

11.1 Research Questions and Answers

This section describes the research questions and their answers. For each of three parts, first the overall research question is answered, which summarizes the main findings, and this is followed by detailed answers to each research question.

Part I is the starting point of this thesis as it analyzed the transitions and the consequences. The overall research question for this part was as follows:

RQ 1: What transitions have taken place in the development of software for consumer electronics products and what are the consequences for building software product lines?

Answers: The software development for consumer electronics has transitioned in the following ways:

1. The software that is embedded in the device is in most cases no longer developed by a single firm, but is developed through a supply chain in which multiple parties develop specialized components for the downstream participants in the chain. The consequences for product line engineering are that components from different suppliers, with their variability, have to be integrated. This causes an increasing effort for system integration and testing; especially because no high degree of modularity has been obtained.

2. Ecosystems are used to support downloadable applications from 3rd parties. In this way organizations transitioned from software product line engineering in an intra-organizational context to an inter-organizational context. Because a successful ecosystem relies on attracting sufficient 3rd party developers, choosing the most suitable ecosystem type and software platform can have major commercial and strategic consequences. The consequences depend heavily on the choice of the ecosystem type.
The research questions that have been addressed in Chapter 2 were:

RQ 1.1: What transitions between industry structures have taken place?

**Answer:** The transition took place in a number of steps: Initially the functionality and the required variability were implemented through hardware components. The second step was that an increasing part of the functionality and variability was implemented through software, but still by vertically integrated firms, meaning that these firms developed the entire product, i.e. both the hardware and software. The third step is that the vertically integrated firms used ICs and software components from specialized suppliers but remained responsible for the overall architecture. The fourth step is that a supply chain approach is formed in which the product is developed by a group of companies and the responsibility of the architecture is shared by the participants. The fifth step is that for supporting 3rd party applications a stable API (Application Programming Interface) has been created and an application store to enable an ecosystem.

RQ 1.2: What were the forces and actors involved?

**Answer:** The research identified 3 types of forces: (A) Forces that are driving, or staying, in a more vertically integrated approach, (B) Forces that favor a more ecosystem-centric approach and (C) Forces, or consequences, that restrain the transition to a more ecosystem-centric approach.

The forces of type A are: (F1) Less efficient use of system resources and therefore higher costs because modularity and a layered architecture introduce inefficiency because no direct control of the hardware is possible. (F2) Reduced options for large innovation steps, because an eco-system centric approach and, to a lesser degree, a supply chain approach, relies on stable interfaces. Large innovation steps often require changes of the high level architecture, which is easier for a vertically integrated company. (F3) The overall quality is more difficult to guarantee because it relies on the quality of components from several suppliers. An individual firm can control this more easily.

The forces of type B are: (F4) Lower development costs and shorter time-to-market, because the development effort can be shared between the participants and because using third party components takes less throughput time then developing these components in-house. (F5) More variability because components from different suppliers can be used and the next party in the supply chain can develop the required functionality more easily.

The forces of type C are: (F6) Increased time for system integration, especially when components using different component technologies, i.e. heterogeneous components, are used. (F7) Higher interaction costs with the component suppliers (F8) The fear that a dominant firm creates a monopoly and takes most of the revenues.

Initially there was only one actor involved, namely a vertically integrated firm that developed and manufactured the ICs, the required software and created the device. After the transitions the following actors are involved: IC manufacturers, middleware vendors, vendors of specialized software components, vendors of operating systems, set makers and application developers.
RQ 1.3: How were the architectural interfaces between the actors defined, and what were the consequences for modularity, variability and product integration?

**Answer:** The architectural interfaces were initially defined by the vertically integrated firms, including the phase in which integrators used specialized suppliers but defined the overall architecture. After the transition, i.e. de-verticalization, to the supply chain industry structure, the interfaces are defined by the participants together.

The research showed that a major challenge lies in providing the right balance between delivering the required variability within the available time-to-market, while retaining sufficient control over the systems architecture to obtain optimal resource utilization. This challenge originates from the nature of consumer electronics that are designed to perform specific tasks using ICs with a low cost price.

The consequences of these transitions are that the effort for product integration has substantially increased because no high degree of modularity was obtained. This relates to the effort to match the interfaces between the supplied and in-house developed components, the effort to remove faults in the software of the supplied components and the effort to remove faults that are caused by unexpected interactions between with the in-house developed components and the supplied components.

For managing the variability, the transition means that components, including their variability from several suppliers and in-house developed components should be integrated, possibly using different component technologies and non-matching interfaces.

The research questions that have been addressed in Chapter 3 were:

**RQ 1.4:** What types of ecosystems are used in consumer electronics products to support 3rd party applications?

**Answer:** Three types of ecosystems are identified that are currently used in the industry to support 3rd party applications. These types are (1) the *Vertically integrated proprietary hardware/software platform* that consists of the hardware, proprietary closed source software and includes the device. (2) the *Proprietary, closed source software platform* that consists of a proprietary closed source software platform but does not include the hardware nor the device and (3) the *Open source software platform* that consists of an open source software platform and also does not include the hardware and the device.

This classification is based on two properties: (A) whether the software is proprietary or open source, and (B) whether the hardware and device are included. Therefore these ecosystems differ in the scope of the platform and the extent to which complementors can modify the software and contribute to the platform.

**RQ 1.5:** Which of the ecosystems types is most suitable for a specific product category from a software engineering perspective?

**Answer:** The most suitable type of ecosystem depends on a number of factors that have been identified: Development costs, quality, variability, speed of innovation, system
Chapter 11

resources, hard real time requirements, stability of the interface and effort for system integration. Note that these factors are very similar to the forces that cause a transition as described in chapter 2.

The most suitable type differs per product type and is based on comparing the strength and challenges of each ecosystem type with the market requirements of a product type:

- When the speed of innovation is high and a low degree is variability is required, the vertically integrated hardware/software type is most suitable. This has been analyzed to be the case for gaming consoles.
- When a high degree of variability is required and it is needed to share the development investment of the software platform between multiple firms, the open source software type is most suitable. This has been analyzed to be the case for digital photo cameras, digital televisions, smart phones and smart watches.
- When stability of the interface is a key requirement, while development costs and quality are also important factors, then the proprietary platform type is most suitable. This has been analyzed to be the case for personal computers.

For tablets, all three ecosystem types are considered equally suitable since these products have characteristics of both smart phone as well as personal computers.

The overall research question for part II is derived from the findings in part I as it identified that supply chains are the dominant structure for software development for consumer electronics and that the time for system integration has increased substantially. Therefore the overall research question for this part was:

RQ 2: What are the consequences for variability management, when applied in software supply chains, and what modelling approaches can support this?

Answers:

- Variability models that are used in software supply chains should be able to support multiple product lines, staged configuration and should support combining the variability from different suppliers. This can be modelled by capturing the variability of the context of a product in a separate tree of the feature model.
- The description of the variability of the components from several suppliers, e.g. through variability models, have to be combined. This can be modelled by using a separate feature model that facilitates the configurations process and separate feature models of the suppliers. In this way the links between the feature models of the supplier’s components and their development artifacts are retained.
- Components from several suppliers and in-house developed components have to be integrated which often have non-matching interfaces and are possibly implemented using different component technologies. In current practice the manual creation of glue components requires a large development effort and frequently leads to faults. An approach using model driven architectures can generate the glue components in an efficient manner while it supports staged configuration. This approach also avoids that a large group of customers and developers need to learn the complexities of model driven architectures.
Conclusions

- Variability models in software supply chains often lead to large models with complex and many dependency relations. A textual variability language can support this more easily than a graphical variability language because when using textual languages, these models can be created and edited more efficiently.

The research question that has been addressed in Chapter 4 was:

**RQ 2.1:** How can feature models be constructed to model multiple product lines in software supply chains?

**Answer:** Feature models can be used by adding a separate tree that describes the context in which a product is used. This context can be a geographical region, a product type or a customer. This method allows for staged configuration in which certain configuration choices are made by an upstream participant in a supply chain, resulting in a partly configured feature model, while the next participant in the chain may perform additional configuration choices. Furthermore, this method allows the merging of feature models by the next participant in the supply chain. This can be done in a stepwise manner by first merging the feature trees, then the context variability trees and finally by adding the constraints that arise from the combination of the features and context variability. The method can be used with commercially available variability management tools and can be applied by the industry without further research needed.

The research question that has been addressed in Chapter 5 was:

**RQ 2.2:** How should feature models be constructed which capture the variability from the components from alternative suppliers, partly offering the same feature set?

**Answer:** When components are used from different suppliers with their variability, this leads to an overall feature model that combines the variability from the different suppliers in addition to the in-house developed variability. When the suppliers offer partly the same feature set this leads to overlapping features, i.e. features that represent alternative implementations.

When this is modelled by using alternative subtrees in the overall feature model, this would lead to a time consuming configuration process to select the desired feature set. An alternative is to merge the feature sets and use a context variability tree to make a selection based on a supplier. A blocking issue here is that feature models do not stand on their own, but are linked to the development artifacts to instantiate a product variant, and therefore merging feature models will not retain the links.

To address the limitations above, a so called *Composite Supplier Feature Model* can be used. This Composite Supplier Feature Model consist of a *Supplier Independent Feature Model* which is linked through dependency relation to the separate *Supplier Specific Feature Models*. This modelling technique allows for an easy configuration process, while the links between the feature models and the development artifacts are maintained. This method, similarly as the method described in chapter 4, can be used with commercially
available variability management tools and can be applied by the industry without further research needed.

The research questions that have been addressed in Chapter 6 were:

RQ 2.3: What are the challenges that arise from combining components with non-matching interfaces and what are the limitations of current practice?

Answer:
The challenge is to create glue components that are able to bridge the difference between the components of different suppliers and in-house developed components. These differences may be caused by the use of different component technologies, which may include different mechanisms for interfacing, binding and implementation of variation points, and by syntactic differences, such as different names of interfaces, methods and parameters.

The challenges that arise from the use of software supply chains are: (1) To protect intellectual property rights it may be necessary to avoid that the next parties in the chain receives source code. (2) For commercial reasons, variation points that are offered to one customer may not be revealed to other customers (3) The various build environments that are used by the suppliers must be supported. (4) The capabilities should be preserved of component technologies that support reachability analysis to exclude unneeded code. The latter is especially important for resource constraint devices, such as consumer electronics.

In current practice three approaches exists for the creation of glue components:

- The first approach is that all possible glue components are created during domain engineering and configured during application engineering. The downside is that this leads to a large development effort during domain engineering and because of the large number of glue components, also a large number of links between the variation points and the glue components are required.
- The second approach is to create the required glue components during application engineering. The downside of this approach is that this will lead to a long throughput time during application engineering.
- The third approach is to use a common standard interface and component technology. The downside is that each pair of components requires two glue components and that these glue components would be very complex since they have to use a common component technology that encompasses all the component technologies that are considered. The creation of such glue components could lead to faults easily.

RQ 2.4: How can MDA be used to bridge mismatches between heterogeneous components in software supply chains?

Answer: The model driven approach (MDA) introduced in this thesis is based on using a supplier independent component model to describe the reference architecture. This reference architecture contains conceptual components, which are components that
Conclusions

contain no implementation and are independent from the component technologies that are used by the several suppliers. Furthermore, it describes the composition between these components in terms of the dependencies between the components as untyped ports, because the actual interfaces are determined by the concrete components of the suppliers.

When suppliers are selected, a model-to-model transformation creates a new model, in which the supplier independent components are substituted by supplier-specific components. This is called the supplier specific component model. At this point, the components of the supplier specific model are tagged with the component technology that is used to implement them.

A second model-to-model transformation generates skeleton glue components. This is done automatically, and only when needed, based on one of the following criteria: (1) when a pair of components, that have connected ports, are tagged with different component technologies (2) when a required interface does not match the provided interface, e.g. when different method names are used or different order and type of their parameters.

Following this step, a combination of model-to-code transformations and reusable code snippets are used to create the required glue code to complete the skeleton glue components. After this step, the build scripts are created. Then the variation points can be configured and the code is created. Following this step, the code can be transferred to the customer, where it remains possible that some of the variation points are configured by the customer or the next participant in the supply chain.

The advantage of this method is that the architecture can be defined independent from the selected suppliers and the selection of the suppliers and the creation of glue can be postponed until the desired decision have to be made, i.e. during application engineering.

RQ 2.5: What are the consequences for the development process and the development roles?

Answer: The process consists of the following main steps: During domain engineering the feature model is created by the requirements manager and the reference architecture by the domain architect. The feature model contains both the functional variation points as well as the so called supplier variation points, which represent the different suppliers. This feature model can be constructed as described in chapter 5.

During application engineering first the supplier is selected by the COTS engineer. Then the glue components are created and configured based on model to text transformations. In this step the COTS engineer is supported by a set of wizards that guides the engineer in specifying the mapping between the provided and required interfaces of the components that need gluing, i.e. for which a skeleton glue component has been created. Here the engineer can select the required snippets from a snippet library or by entering the code manually, which is needed when a new component technology is used.

Then the customer support engineer liaises with the customer to determine what configuration is required and generate the build scripts. The partly configured product is delivered to the customer who can make additional configuration choices using a variability management tool.

227
Chapter 11

For our approach a number of different roles are identified and the specific skills that they require:

- There is the so called meta-team, which consists of experts in model driven architectures that create the UML meta-model, the transformations and maintain the templates for the transformations.
- The domain architect requires working knowledge of MDA to create the reference architecture.
- The requirements manager requires working knowledge of feature modelling to create the feature model.
- The COTS engineer uses the MDA tool with the wizards and snippets library to create the glue components. For these tasks no knowledge of MDA is required, but knowledge of different component technologies is required.
- The customer support engineer uses the MDA tool to generate the components for the customer and uses a variability management tool to make configuration choices. Also the customer support engineer does not require knowledge of MDA.
- The customer can make the final configuration choices using a variability management tool.

The research questions that have been addressed in Chapter 7 were:

**RQ 2.6:** What are the benefits of TVL for modeling product line variability, as perceived by model designers?

**Answer:** TVL has the following advantages: (1) It is easier to interpret large models in comparison with graphical languages. (2) The learning curve is moderate, especially for developers who are familiar with textual languages such as C, C++ or Java. (3) Editing and creating a model is easier as it involves copying and pasting of plain ASCII text, whereas with a graphical language often many mouse clicks, drop-down menus etc. have to be used. (4) Constraints between features can be captured more easily than with a graphical language and the feature model remains easy to interpret with a large number of constraints. (5) Feature attributes are an explicit part of the model instead of an add-on. As a result, constraints involving attributes can be described easier.

**RQ 2.7:** What are the product line variability modeling requirements that are not fulfilled by TVL?

**Answer:** The following capabilities were missing in TVL: (1) Attribute cardinalities. (2) Feature cloning. (3) The possibility to capture the default value, meaning that during configuration many choices have to be made, although many may be standard choices. (4) The possibility to capture error and guidance messages that can be used during product configuration.

The overall research question for part II is derived from the findings in part I and part II as it is identified that there is an increasing amount of faults in the software and an increasing
pressure on time-to-market which results that there is less time available for testing. Therefore the overall research question for this part was:

**RQ 3:** How can test efficiency be improved when the risks and user product interaction are taken into account?

**Answers:**

- The test efficiency can be improved when operational profiles are used to guide the test effort. When a very high reliability is required, meaning that the aim is to remove all faults, it is more efficient to focus more test effort to those development artifacts that contain the most faults.
- Risk based testing can be improved by allocating more test cases to operations that have a high frequency of use, which can be identified by using three dimension to capture the risks, instead of two. This is an improvement of current practice in which relatively too much test time is given to operations that have a higher probability of causing a failure.
- Risk based testing can be applied in software product line engineering by capturing the percentage of use of the development artifact in the set of variants. Based on the percentage of use and the criticality of each development artefact, a selection is made of which development artefacts should be tested during domain engineering and for which testing should be deferred to application engineering.
- For reducing non-technical failures, caused by an uncertain user-product interaction, the tests should include the differences in types of users, their use-process and the environment in which a product is used.

The research questions that have been addressed in Chapter 8 were:

**RQ 3.1:** What are the benefits for the test efficiency when using operational profiles?

**Answer:** The reliability of a product is a function of the number of faults in the development artifacts that implement the operations and the chance that these faults lead to a failure. When operational profiles are used, most test cases are allocated to the operations that are used most frequently. As a consequence most faults will be found in the development artifacts that are used most frequently and therefore the reliability increases most of those operations that are implemented by these development artifacts. Since the overall reliability is a weighted sum of the reliability of the individual operations, the reliability will increase more when the most used operations are tested more heavily.

However, this effect disappears when an artefacts that implements an operation does not contain faults anymore. Using more test cases for these operations will not increase the total reliability anymore. In that case, which means that the system contains hardly any remaining faults and therefore has an extremely high reliability, it is better to allocate more test time to those operations that still cause failures. Therefore the conclusion is, which is supported by a statistical analysis, that operational profiles increase the test efficiency, however not when a very high reliability is required.
**Chapter 11**

**RQ 3.2:** Which dimensions in risk based testing require most attention to reduce the overall risks?

**Answer:** In the current definition and practice of risk based testing, a risk matrix is used which consists of two dimensions: The *probability dimension* and the *impact dimension*. The probability dimension is based on the technological complexity; whether the software is develop by inexperienced developers and whether it contained many faults in the past. These aspects are also used in defect (fault) density predictions and therefore the probability dimension indicates how many faults are expected in a certain area. The impact dimension is based on a number of aspects, such as the visibility of the operation to the end-users, the impact on the business when it fails and the frequency of use of an operation.

When the risks are captured in a formula, this will be a product of three aspects, namely: The *chance that an operation fails* multiplied by the *change that an operation is executed* multiplied by the *damage that a failure might cause*. In other words, the impact dimension should be separated in two dimensions, namely the *change that an operation is executed*, which is determined by the frequency of use, i.e. the operation profile, and the *damage that a failure causes*. It can therefore be concluded, which is supported by a statistical analysis, that most attention should be paid to the impact dimension by making the *frequency of use* a separate dimension, which will lead to a risk matrix consisting of three dimensions.

**The research questions that have been addressed in Chapter 9 were:**

**RQ 3.3:** How can risk based testing by applied in software product line engineering?

**Answer:** Risk based testing can be applied in software product line engineering by identifying which development artefacts should be tested during domain engineering and which during application engineering. The identification of these development artefacts is based on the percentage of use of the features that these development artefacts implement and the criticality of the development artefacts.

**RQ 3.4:** What are the consequences for testing during domain and application engineering?

**Answer:** The consequences for testing during domain engineering depend on the testing strategy that is chosen:

- For the *division of responsibilities strategy*, the components and variation points that are used in many products should be tested during domain engineering, especially those that have a high probability and implement features with high impact. The components that are used in a small percentage of products and have little probability and impact should not be tested during domain engineering.
- For the *test assets for reuse strategy*, only for those components and variation points that are used in many products should reusable test assets be created.
Conclusions

- For the sample application strategy, the components and variation points that are used in many products should be included in the sample application. For testing during application engineering this approach has little consequences. It should only be identified which components are already tested during domain engineering and to what degree. For instance, those components that have already been tested intensively during domain engineering should not be tested heavily anymore during application engineering.

**RQ 3.5: What tool support is required?**

*Answer:* To capture the risks and identify the most critical components, the novel concept of quantified feature models has been introduced, which are feature models in which each feature is associated with a percentage of use in the set of variants, and the impact when a feature fails. Furthermore it should be able to model the dependencies between a feature model and the development artefacts, where the percentage of a development artefact, such as a component, can be derived from the features it implements.

A complexity in quantified feature models is that the percentage of a feature usage is related to the percentage of usage of another feature. For instance when one feature requires another feature, than the feature that is required should have a larger percentage than the feature that requires the other feature. Such relations may lead to an inconsistent quantified feature model, for which there are no solutions know in literature that covers the wide range of possible dependency relations and therefore requires further research. Further research is also needed to identify whether such consistency checking is really required, since risk based testing is based on estimations rather than rigorous statistics and therefore a precise consistency checking may not provide additional value to practitioners.

The research questions that have been addressed in Chapter 10 were:

**RQ 3.6:** To what extent can an operational profile be applied to consumer electronics products to model (unexpected) product use for highly innovative consumer electronics products?

*Answer:* The operational profile has two purposes: First, it guides the software test effort and secondly it is used to determine the failure intensity of a product that is caused by technical software failures, i.e. failures where the behavior of the system deviates from the specified behavior. For highly innovative consumer electronics products, the non-technical failures are increasingly becoming the major source of reliability problems. Non-technical failures are caused a by a mismatch between the requirements of the product and the customer’s expectation and because the product is used in different ways than anticipated by the manufacturer. These non-technical failures are not addressed when tests are executed according to the operational profiles. Therefore an operational profile can be used to guide the test effort for software testing, but cannot be used to determine the overall reliability of a highly innovative consumer electronics product and to identify non-technical failures.
Chapter 11

RQ 3.7: What adjustments need to be made?

**Answer:** To identify the main causes of the increasing reliability problems, the experiments have shown that the following factors have a significant impact on non-technical failures and should be included when the reliability of a product is determined:

- **Users:** Different users may have different technical knowledge and experience with using innovative products.

- **Use process:** A product is used in a different manner at different moments of time. Typically the following phases can be recognized: Unpacking, installation, configuration, first use, extended use, end of use. It has been identified that in the early phases most reliability problems are encountered.

A third factor was analyzed, namely the **Environment:** A consumer electronics product is often connected to other products, for instance a multimedia player is connected to a television, the internet and an amplifier. Often failures occur when a device cannot be connected properly to another device, for instance because the connections are not available. However the experiments did not show a clear effect of the influence of the environment on the product’s reliability.

11.2 Threats to Validity

In this section the threats to validity of the results are discussed using the validity threat types that are proposed in [Feldt 2010]. Since each chapter has its own research results, the threats are discussed for each chapter separately.

A threat to external validity of chapter 2 is whether the conclusions, based on two case studies, are generally applicable to consumer electronics industry. These threats were mitigated by using two different case studies and two different data collection methods. A specific threat is that the results may not be applicable to consumer electronics that have a much smaller size of software or for products for which there is less pressure on cost price and time-to-market. For this chapter, the author sees few threats to internal validity: For the case study of digital televisions multiple sources of data were used and the authors had detailed insight as they were involved in this product for many years. For the case study of mobile phones, also the authors had experience with the software development for these products types and many studies are available in related art.

The case studies in chapter 3 cover a wide range of consumer electronics products which gives a high confidence to external validity. A threat to internal validity is that the importance of the factors for each product type are determined by the authors, and not based on a detailed analysis of the market. A threat to external validity is that the strengths and challenges of each ecosystem type are based on qualitative data, mainly from the domain of mobile devices. More quantitative data, preferably from more product types would give stronger evidence. A threat to construct validity is that the factors that determine the most suitable type of ecosystem are based on the strengths and weaknesses form a software engineering perspective and do not include commercial and financial aspects. Therefore the conclusion that the ecosystem type that is used in practice is, in most cases, the same as that which follows from the analysis, could be caused by these factors as well.
Conclusions

For chapter 4 and 5, the author sees few threats to both internal and external validity, since the proposed techniques of creating feature models have been exercised on concrete cases and demonstrated with a number of scenarios. An internal threat to validity is that merging of feature models may reveal unexpected problems, because this could not be exercised with the available tools.

For chapter 6 the author sees few threats to internal validity, since the method to use MDA has been implemented and exercised using a concrete case study. A threat to external validity is whether the proposed method is applicable to other component technologies as well, since the case study was only exercised with two component technologies. A second threat to external validity is whether the proposed method does indeed provide the efficiency gains as discussed, since a quantitative comparison was not made between the proposed new approach and current practices.

For the evaluation of TVL in chapter 7, an internal threat to validity is that the models were created by the researchers and not by the participants, which may influence the evaluation of ease of use and costs of model creation. Another threat to internal validity is that the models that have been used were relatively small in size. In general, the wide range of case studies and difference in backgrounds of the participants, give a high confidence in the validity of the results.

An internal threat to validity of the statistical analysis in chapter 8 is that the conclusions are based on data from a single case study and the experiments were not based on real life data. Therefore the conclusions might not be generalizable. The mitigation strategy was to exercise the statistical analysis on a wide range of variables to verify the conclusions. Furthermore, the main conclusions are supported by a reasoning that explains why the statistical experiments shows the results as they are. Another external threat to validity is that the claim that the model is easy to understand, is only supported by a single case study.

A threat to external validity on the use of risk based testing in chapter 9 is that it has been evaluated using a single case study. A specific concern is that the organization, in which this case study has been applied, uses a strict separation between domain and application engineering, which may not be the case in other organizations.

A threat to external validity of the evaluation of the user-product interaction framework is that the tests have been executed on a single product and therefore may not be generalizable for other consumer electronics products. A threat to construct validity is that other factors, besides users, use-process and environment, may influence the results.

11.3 Generalization

Most of the research in this thesis was focused on consumer electronics, where some results are more widely applicable, as shown in Section 1.7. In this section it is discussed whether the results, classified in that overview as applicable to consumer electronics or embedded systems, are also more widely applicable.

Consumer electronics are characterized by a fast increasing amount of software, a high degree of innovation, the lack of standardization and the need to optimally make use of system resources. These are the main reasons why the transitions have occurred as described
Chapter 11

in chapter 2 and 3. Many of these characteristics can be found in other domains as well, especially for embedded devices. It may therefore be expected that the research results and the identified challenges are also applicable to the wider domain of embedded systems. It has already been identified that most of the research results of part II are applicable to embedded system, which is in line with this observation.

The automotive domain is an example in which the functionality, previously implemented by mechanical and electrical components, is being replaced by software. Furthermore new functionality, such as driving assistance and ultimately, self-driving cars, are largely implemented through software. Another example is the medical domain. In this domain increasingly healthcare products are used outside the hospital and are being used in our private homes, which results in cross-over products between consumer electronics and healthcare products. A difference however between these domains and the consumer electronics domain is that failures may cause health and safety risks and therefore the reliability requirements are much higher. Consequently more time has to be spent on ensuring the quality which includes rigorous testing.

The methods that were developed for feature modelling in supply chains, in chapter 4 and 5, were dedicated to be used in embedded systems, because in embedded systems only those features that are required by the market are usually embedded in a product. However these approaches might be also applicable to software product line engineering for software supply chains in general as more and more users get overloaded with features and therefore reducing the feature set may become an explicit goal. Furthermore, this method may also be applicable outside the area of software engineering and possible also for electrical engineering, hardware development and mechanical engineering, since feature modelling itself may also be applicable to these domains. An example was already shown in chapter 7, in which the use of a textual variability language was evaluated using an example from hardware development. A second example is mechanical development in the automotive domain, in which there are also a large number of variation points that have to be managed during the design and manufacturing process.

The method described in chapter 5 for combining overlapping feature models showed the limitations of merging feature models because the links to the development artifacts would be severed and therefore a merged feature models cannot be used to configure a variant. In the method described in this chapter a separate feature model is constructed that allows for an easy configuration with links to the original feature models. This method may therefore also be applicable to merging feature models in general.

The approach in chapter 6 to generate glue components was specifically aimed at embedded systems, however part of this method may also be used in other domains where glue components are required, especially where different component technologies are used.

A specific goal of the research in chapter 6 also was to develop methods that would not expose unfamiliar technology to a wide group of developers. This analysis method may very well be applicable to other technologies as well, since getting familiar with new languages and technologies may take a lot of time and resources, which does not directly benefit the profitability of a company. In the evaluation of the MDA, we identified the roles and their tasks. These roles and their tasks may very well be applicable to the deployment of MDA in general.
Quantified feature models, as introduced in chapter 9, could potentially also be used during commonality/variability analysis to reduce the amount of variation points. Since software product line can easily incorporate thousands of features and variation points, the effort for developing, testing and maintaining a product line becomes problematic in terms of investments [Savolainen 2009]. By identifying how much each feature is actually going to be used, a better informed decision can be taken. For instance features that are never used, or hardly ever, can be removed, thereby reducing maintenance costs. Features that are used in a large majority of the products may become a mandatory feature which makes the configuration process easier and the code can be simplified.

The enhanced framework for user product interaction for consumer electronics product, as introduced in chapter 10, may also be applicable to other domains where non-technical failures are increasing. In many products that contain a large amount of software, the installation and use of a product is becoming more and more complicated and therefore also for these domains the users, use-process and environment are likely to have an effect on the reliability.

**11.4 Key Contributions of this Thesis**

This thesis showed that the increasing amount of variability and speed of innovation, together with a high pressure on development costs and product quality has a significant impact on many aspects on software product line engineering for consumer electronics.

To start with, the impact on industry structures was analyzed, which in his turn had impact on variability management and the transitions created additional pressure on the available time for software testing. For each of these areas this thesis has contributed through a better understanding of the challenges and provided solutions for some of these challenges.

The specific contributions of this thesis for industry structures of software development for consumer electronics are:

- A model is introduced which captures the evolution of software development for consumer electronics and the forces that govern this transition. Detailed case studies are described that can serve as lessons learned for other domains.
- The transition on the consumer electronics industry was compared with that on the computer industry in which the de-verticalization was enabled by a high degree of modularization. This thesis showed that the de-verticalization of the consumer electronics industry started when functionality outside the traditional domain was introduced, despite a low degree of modularity.
- A first step towards decision support method is introduced that determines which type of ecosystem is most suitable to support 3rd party applications for consumer electronics product from a software engineering perspective.
- The architectural challenges are identified that arise when a company considers a different industry structure and when it decides to adopt an ecosystem type to enable the use of 3rd party applications.
Chapter 11

The specific contributions of this thesis for managing the variability in a software supply chain are:

- The concept of context variability was introduced which provides a solution for modeling multiple product lines in software supply chains and this method supports staged configuration. Because our approach is intuitive and based on conventional feature modeling this approach can be easily applied in the industry.
- It was shown that merging feature models is not a most suitable approach to combine variability since it would sever the link to the development artefacts.
- A method was introduced to model the variability of alternative components with overlapping features. An advantage of this approach is that the original feature models have been kept “intact”, meaning that no changes have to be made to the feature models of the supplier’s components. Furthermore, this approach can be used with existing variability management tools and the case study showed that this approach is straightforward to use.
- A model-driven approach is introduced for automating the integration of heterogeneous components, covering syntactic mismatches and mismatches related to different component technologies. In this approach glue components are generated efficiently only when they are required, staged configuration is supported, and the additional skills required to deploy MDA are localized in the organization which ensures that only a limited group of developers are exposed to unfamiliar technology.
- A textual variability language, TVL, was evaluated through an empirical evaluation on four product lines and developers from the industry. This showed that using TVL can lead to an efficiency gain, especially when using large feature models with complex constraints and feature attributes. The evaluation also revealed that some capabilities were missing, such as attribute cardinality, default values and guidance messages for product configuration.

The specific contributions of this thesis for software testing are:

- A statistical model is introduced that captures the relation between faults and failures. Using the model the efficiency improvements of using operational profiles are determined and an improved method for risk based testing is presented. The model is easy to understand which gives it great explanatory power in illustrating the influence of different amounts of test cases on the product’s reliability.
- An approach is introduced to apply risk based testing in software product line engineering. This approach can be applied with different test strategies and addresses the disadvantages of these strategies.
- Quantified feature modelling is introduced, which can support risk based testing for software product line engineering and which may, given further research, also lead to an improved commonality/variability analysis.
- The limitations are identified of using operational profiles for highly innovative consumer electronics products. An enhanced framework is developed to structurally analyze unexpected user–product interactions for highly innovative consumer electronics products. This framework can be used during the test process to reduce both technical and non-technical failures.
Conclusions

The results in this thesis can be used by the industry as part of their strategy, development and test process as follows:

- The models of industry structures and ecosystem types can serve the decision making process for individual firms for selecting the most appropriate industry structure for their business and ecosystem type when a transition is considered. Furthermore these firms can proactively address the challenges that are identified.
- The methods that are described for variability management can be used to improve the practice of software product line engineering when applied in a software supply chain. These methods are widely applicable to the domain of embedded software and, for some of the described methods, to software product line engineering in general.
- The methods introduced in this thesis to improve the test efficiency and to identify non-technical failures can be used by the industry to improve the reliability of their products while preventing a longer time-to-market.

11.5 Further Research

In Section 11.3 the generalization of the research was presented which mostly result in further research. In this section further research is discussed which is not covered in that section. This section covers further research that are directly related to the individual research results as well as identifies topics for further research for software product line engineering for consumer electronics in general.

Evolving industry structures

The model of industry structures shown in chapter 2, captured the system as a whole and captured the role that modularity plays in the transitions. When systems become larger, it may be expected that for some parts of the system a high degree of modularity can be obtained, e.g. between the middleware layer and the application layer, while for other parts of the system, e.g. between the middleware and firmware layer, this might not be obtained. This leads to a more fine-grained, possibly hierarchical model. Further research in this area is needed to identify in which parts of the system, interfaces can or cannot be kept stable and what the consequences are for other parts of the system.

A principal area of further research is what the consequences are for the architectures and development process when an existing application framework is integrated that is used for a stable interface for 3rd party applications. As shown in chapter 2, integrating existing application frameworks, sometimes referred to as middleware, is already common practice. However in that situation the applications were integrated into the device and the combination could be tested prior to delivery to the customers. In the situation when an ecosystem centric approach is adopted, it is not a priori known which applications are downloaded by the end-users and what the consequences for system utilization are, which makes it more difficult to guarantee the correct functioning and performance of the product.

Also more research is needed to identify the benefits of a mixed model of ecosystems for consumer electronics in which part of the software platform is open source, while another
Chapter 11

part is closed proprietary. Such an approach might provide the benefits of both these ecosystem type, e.g. by offering a stable interface towards the applications, based on proprietary standards, while an open source approach with open standards are used to support different hardware configurations.

Deployment of MDA in software supply chains

One of the areas of further research to create glue components, as described in chapter 6, is to support the capabilities of some component technologies that support reachable analysis to avoid the use of unneeded code. Also further research is needed to generate glue components that bridge larger semantic differences, which requires a more general model of glue code.

The research on MDA showed that different participants in the supply chain use different incompatible component technologies. To transfer the software to the next participant in the supply chain, the proposed solution in this thesis is to transfer standard C code and build environments. Furthermore, also feature models are delivered to the customer, which assumes that these models can also be used by the customer. As another example consider MDA to create code, i.e. not only glue code. To optimally use this technology, both the supplier and customer have use it, so that models can be transferred, rather than code. The receiving party could then integrate and test the models and generate the code once the final configuration choices are made.

These examples reveal a general problem in software supply chains, namely that different participants use different technologies and different tools, which may be incompatible. The consequence is that the introduction of new technology is restrained because it can only be optimally used when the other participant in the chain have also introduced this new technology. Further research is needed how technologies can be standardized across a supply chain, or how to deal with incompatibilities in technologies and tools.

Ensuring sufficient quality in a supply chains and ecosystem centric approach

Currently, multiple ecosystems are used for similar products, e.g. mobile phones or digital televisions. For application developers this means that they have to develop and test their applications on a number of different platforms. This is particularly challenging when open source software platforms are used as this often this leads to fragmentation. The results of this thesis may support the application developers in three ways: (1) Context variability, as presented in this thesis in chapter 4, can be used to capture what the variability in the platforms is for which the application is going to be developed. (2) A quantification of the context that captures the market share, similar as presented in chapter 9, can be used to determine which platforms should be targeted first (3) This quantification can be used to set priorities for testing.

Another problem that requires more research is that components or applications from 3rd parties may use different user interface styles. A concrete example from a decade ago was a product that combined a DVD-player with a video recorder. This product contained two styles of user interfaces. Because the product was created from components from different
suppliers and the user interface was tightly coupled with the middleware, it was not possible within a reasonable period of time, to create a product with a consistent user interface. A similar problem occurs when applications are used from different suppliers. It frequently happens in mobile devices that one application uses a physical “settings” button, another application uses settings from within the application, while a third uses a separate application to control settings.

When risk based testing is applied in software product lines, an upstream participant may not know the criticality and the percentage of use of a development artefact because that is typically only known at system level. Therefore an upstream participant is not able to determine how rigorous each development artefact needs to be tested. This may result in a situation in which to some artefacts too much test time is allocated, and worse, to some artefacts too little test time. An approach needs to be developed in which the upstream participant receives this information from the downstream participants.