Chapter 1
Collaborative Development of Embedded Systems

Marcel Verhoef, Kenneth Pierce, Carl Gamble, and Jan Broenink

1.1 Introduction

This chapter motivates our interest in collaborative modelling for embedded systems design by describing the challenges faced by developers of contemporary embedded systems. We set the scene in Sect. 1.2 indicating how ubiquitous embedded control systems are in our daily life, and identifying the issues faced by industry when producing solutions of sufficient quality in a timely manner. In Sect. 1.3, we list the main challenges that we see in the development of this kind of product. In Sect. 1.4, we introduce a small fictional case and investigate it from the perspective of the different disciplines, illustrating the motivation for collaborative development. In Sect. 1.5, we explain how the Crescendo technology can address the challenges posed by embedded systems design. Section 1.6 concludes the chapter with a small summary.

M. Verhoef (✉)
Chess WISE, Haarlem, The Netherlands
e-mail: Marcel.Verhoef@chess.nl

C. Gamble • K. Pierce
Newcastle University, Newcastle upon Tyne, UK
e-mail: carl.gamble@newcastle.ac.uk; kenneth.pierce@newcastle.ac.uk

J. Broenink
University of Twente, Enschede, The Netherlands
e-mail: J.F.Broenink@utwente.nl

1.2 Setting the Scene

Computers are all around us and we use them every day, sometimes even without giving it a second thought. The term "computer" often refers to the Personal Computer (PC), which is used to send e-mail and browse the Internet or perhaps a video game console that is used for entertainment. But computers are also part of the alarm clock, coffee machine, dishwasher, video recorder, DVD player, camera, television set and mobile telephone. This class of systems is often referred to as "embedded systems" and you can easily count up to a hundred embedded devices in an average family household nowadays [25].

We become more and more dependent on the proper operation of these embedded systems. Not only because they are efficient and convenient to use but also because they potentially affect the quality of life. Sangiovanni-Vincentelli already mentioned in his key-note presentation [88] at the 2006 Design Automation and Test in Europe (DATE) conference that a modern, high-end car contains 80 microprocessors executing several million lines of code and this trend has continued to grow over the last decade. These microprocessors are used to control not only the car radio and air conditioning, but also the air bag, cruise control, fuel injection, brakes and power steering. A failure in any one of those critical embedded systems may have severe consequences. But the general public is typically not aware of this because these computers are deeply embedded in the system, hidden well out of plain sight. Dependability issues are typically associated with the military, medical or aeronautical domains but not so much with consumer or capital goods. For example, no one asks about the code coverage statistics of the power steering unit (an embedded system that contains a microprocessor which executes possibly several thousand lines of code) when you buy a new car. In 2004, Deutsche Welle reported[1] that the reliability rating of German cars, which used to be unrivalled and universally acclaimed, has been steadily decreasing for several years in succession as compared to their main competitors. Analysts believe that this may very well be due to the increased complexity as outlined by Sangiovanni-Vincentelli.

The economic relevance of embedded systems is easily demonstrated. For example, take mobile telephony. Market analysts such as Informa Telecoms & Media[2] predicted in 2005 that the number of mobile handsets deployed worldwide would reach one billion early in 2007 which corresponds to roughly 15 % of the population on Earth. Moreover, this target was reached in just 15 years and the market is far from saturated. Growth is continuing, at the time of writing there are an estimated 6.8 billion active mobile phone accounts.[3]

These numbers are just staggering, and it is obvious that such a market potential generates an enormous amount of pressure on the companies that build these kinds

[1]See http://www.dw-world.de/dw/article/0,2144,1400331,00.html.

[2]See http://www.informatm.com/.

[3]See `ICTFactsFigures2013.pdf` from http://www.itu.int/.

of products. Production volumes are extremely high, profit margins are typically low which implies that you have to reach the market with a new product before your competitor, in order to be economically successful. This so-called “time-to-market” (TTM) pressure is therefore the beast to beat. Companies like Nokia and HTC were mobile handset market leaders in 2010, but their marketshare has dropped spectacularly since Apple and Samsung introduced their smartphones. This also caused a spectacular shift in the market for software ecosystems. Microsoft ruled the personal computers era with Windows for two decades, but is now struggling to keep its marketshare in mobile computing (for tablets, smartphones) as they are competing against iOS (Apple), Android (Google) and Linux.

Companies invest huge amounts of money and effort in order to reduce the production time and cost price of their products. This has created a secondary economy consisting of companies that deliver (half-) products and services to achieve those goals. For example, Gartner[4] reports that the revenues for Electronic Design Automation (EDA) experienced double-digit growth in 2006, reaching 4.5 billion US dollars. Over the last decade, the market has stabilised to 5.7 billion US dollars per year (2012 figures), but this stabilisation is mainly due to the economic downturn that started in 2010, which caused investments to be postponed. But do all these investments lead to good products? Unfortunately not. It seems that the well-known adage “Price, Time, Quality - Pick Any Two for Success”, as is shown in Fig. 1.1, is still a fact of life.

After the famous CHAOS report from the Standish Group[5] appeared in 1994, there have been numerous published examples of projects failing or products malfunctioning [49]. Although the way these numbers have been gathered and reported has been criticised by the academic community [33], the same trends have also been independently demonstrated by, for example, the Software Improvement Group and Coverity. The latter company produces a yearly report in which they publish measured software defect rates of open source software [26], including the Linux operating system, the Network Time Protocol (NTP) and the Xbox media center (XBMC). Coverity analysed 380 million lines of code from 250 proprietary code bases, with an average code base size of 1.5 million lines of code. The average defect density (of high- and medium risks) measured was 0.69 per thousand lines of code. The trend since 2009 was that defect density is going up rather than down, not because of deteriorated quality of the code analysed but due to the improved performance of the static analysis tools used! In 2012, roughly 5,800 defects were detected in the 7.4 million line Linux code base and approximately 5,200 of those were fixed within the same year. Since 2006, 15,000 defects were found in total and to date some 8,400 are fixed. Despite these numbers, Linux is considered to be one of the most robust pieces of software as the mindset of the developers is to continuously improve the code base by using static analysis tools, among others.

[4]See http://www.gartner.com/, Doc. Id. G00143619.

[5]See http://www.projectsmart.co.uk/docs/chaos-report.pdf.

We offer three kinds of service:
GOOD - CHEAP - FAST
You can pick any two
GOOD service CHEAP won't be FAST
GOOD service FAST won't be CHEAP
FAST service CHEAP won't be GOOD

Fig. 1.1 Decision making at large: how to find the optimum?

Despite efforts to improve the quality of computerised systems, it remains difficult to make error-free systems, as the previous example demonstrates. Most surprisingly, end-users seem to have accepted that as a given fact. People are used to reboot their computer if a problem occurs. If it does not work, you just download the latest software from the web site. Updates and upgrades have become part of the business model of the product. Even more so, only limited warranties[6] are provided and companies typically do not accept any liability from the use of their products. Would you buy a car if you would have to sign such a legal document? Open source software comes with a so-called *"as-is"* disclaimer, without warranty of any kind. The GNU General Public License[7] actually contains the following sentence: *"The entire risk as to the quality and performance of the program is with you."* Yet, open source software is often believed to be of higher quality than most commercial software because it is exposed to public scrutiny.

Quality is a major issue in embedded systems development, mainly because of the production volumes involved. Intel Corporation was forced to recall a substantial number of their early Pentium processors in 1994 because a problem was found in the floating point unit after the product release. Harrison reported at the 2005 ForTIA Industry Day [71] that Intel wrote off 475 million dollars because of the Pentium FDIV bug and suffered considerable damage to their reputation. But even in a low-volume market things can go spectacularly wrong with great consequences. On June 4, 1996, the inaugural flight of the Ariane-5 rocket failed. About 40 s

[6]See for example the End-User Licence Agreement at http://www.microsoft.com/.

[7]See http://gplv3.fsf.org/.

after initiation of the flight sequence, at an altitude of about 3,700 m, the launcher veered off its flight path, broke up and exploded. The Cluster mission, consisting of four identical scientific satellites, was lost during this event. Conservative estimates suggest that this accident costed the European tax payer in the order of 300 million Euro. The Inertial Reference System (abbreviated in French: SRI), which is used to determine the attitude of the launcher, shut down mid flight because an exception occurred in the software calculating the current flight path. Virtually the same system had been used to launch Ariane-4 rockets successfully for many years, but it was used outside its original specification in this particular case. The investigation showed that the system was never tested under flight conditions despite suggestions from the responsible engineers. In fact, the Ariane-5 Accident Report [67] states: *"… it was jointly agreed* not *to include the Ariane-5 trajectory data in the SRI requirements and specification."* However, the report does *not* state *why* this decision was made. It is commonly believed that the TTM pressure, to have this new generation launcher operational as soon as possible, may have contributed to this decision, taking into account the excellent track-record of a similar system on Ariane-4. Johnson reports on similar problems at NASA in [48]. The "Faster, Better, Cheaper" initiative, which was announced in 1998, fostered a culture in which engineers took considerable risks to innovate with new designs in order to meet requirements. In hindsight, TTM is one of the contributing factors [51] to the loss of the Mars Polar Lander mission in 1999.

But the tide of acceptance seems to be turning, in particular in the area of security. Here, users are less willing to accept failure in applications where trustworthiness and reliability are paramount, such as authentication, digital identity, privacy and on-line payments. In combination with the explosive growth of the Internet, a myriad of exploits have been reported due to software vulnerabilities in browsers, for example. An extensive overview of these can be found in the digest of the Usenet newsgroup `comp.risks`. An entire industry, lead by Symantec and McAfee, has grown around providing anti-virus and anti-malware scanners and services. These issues seemed to be isolated to the personal computer, but now it is also entering the realm of embedded systems. In 2008, the Digital Security group at Radboud University Nijmegen demonstrated the feasibility of breaking the cypher used on the Dutch public transport NFC payment card, in real time, by eavesdropping on the encrypted communication between the card and the terminal.[8] This enabled them, with very simple means, to copy the contents of the NFC card on the fly so that one could in principle travel for free. In 2013, the same research group demonstrated that it was possible to use a similar strategy to hack the secure wireless connection for remote controlled car keys, not only giving you access to the car, but also the ability to start it. Both examples attracted a lot of media attention[9] but despite the public outcry, instead of fixing the problem, the researchers were ordered to refrain from publishing their results. The real problem here is that it cannot be fixed

[8] See http://www.ru.nl/ds/research/rfid/.

[9] See, for example, http://www.bbc.co.uk/news/technology-23487928/.

by a mere software upgrade as it requires replacement of (potentially millions of) physical devices in the field. It is fair to say that the latter two examples are not related to software problems, but are a case whereby a priori assumptions about the robustness of certain cryptographic solutions are overestimated against the ever-increasing power of modern computers. Nevertheless, with the advent of software only solutions, such as openssl[10] and bitcoin,[11] one could imagine the economical impact in case these could be broken due to software errors.

1.3 The Embedded Systems Design Challenge

One might argue that the examples mentioned above are dated and do not reflect the current state of practice. But ongoing research, such as [26, 49], has shown that the average likelihood of projects succeeding has only *marginally improved* over the last decades despite substantial investments in tools and processes. It is generally believed that the performance in the embedded systems domain is rather worse than better. Why is this the case? Looking at general trends there are a few potential reasons.

The design gap problem. According to Moore's law [75], the performance of hardware is roughly doubled every 18 months. But recent advances in networking, packaging and integration technology have enabled the development of heterogeneous embedded computing platforms that show a potential exponential growth in performance and thus complexity.[12] These platforms are commonly referred to as System-On-Chip or Network-On-Chip and usually combine multiple and interconnected radio-frequency, analog and digital components on a single chip. However, the technology we use to design the applications for these new platforms cannot keep up with this tremendous growth in capabilities, primarily because they are currently focused on designing single, monolithic systems. In other words, the complexity of the problem grows much faster than the capabilities of today's leading design tools. This is commonly referred to as the "design gap."

The moving target problem. Rapidly evolving technology and the constant quest for reducing cost-price forces designers of embedded systems to operate on the edge of what is technically feasible. In order to stay competitive, they sometimes need to adopt novel technology even while a product is already under development. One of the key problems in embedded systems design is the *validation* of these design decisions. How much effort and time does it take to check that the intent of a design choice works out well in practice? Over-dimensioning is the usual approach to accommodate for uncertainty in the design, but this is typically not economically

[10] See http://www.openssl.org/.

[11] See http://bitcoin.org/.

[12] International Technology Roadmap for Semiconductors, see http://public.itrs.net/.

viable because it usually increases the cost price. Sometimes actual prototypes need to be built in order to assess the feasibility of some potential solution. Managing this process is regarded as the key to success, and it is often referred to as "shooting at a moving target."

The requirement versus design paradox. Making design decisions in the early phases of the system life-cycle is notoriously difficult. In this stage, requirements are often unclear and under-specified, at best leading to a long list of properties that the system shall eventually satisfy. In the past, emphasis has been put on managing the requirements process, such that sufficient information is available at the time the design decisions are made. However, this is often not realistic, in particular in the domain of embedded systems. At the time when requirements are elaborated, the major architectural design decisions also need to be taken, primarily in order to meet the TTM target for the product. But how can one make these crucial decisions when there is still so much uncertainty? This is in particular true for performance criteria that the system must meet because they are in general surprisingly hard to quantify and evaluate. It is obvious that elaboration of the requirements is guided by the chosen architecture but in turn the definition of the architecture depends on clear and unambiguous requirements. System architects have to deal with this paradox, for example, by applying iterative development processes in order to close the design loop.

Late closure of the design loop. Distributed real-time embedded control systems are inherently complex and so is the associated design process. Many implementation choices need to be made, and the impact of each decision is difficult to assess due to this complexity. This makes the design process error prone and vulnerable to failure as other downstream design choices may be based on it, causing a cascade of potential problems. Moreover, it may take some time to realise that a decision is wrong because it requires feedback in the design process. Usually this happens late in the life-cycle, at system integration and testing or even product manufacturing. The repairs required to fix these problems cause significant project delays and cost overruns or sometimes even worse: product cancelation.

Multidisciplinary design. Systems are traditionally designed in a mono-disciplinary style usually with an organisational structure to reflect this (e.g. mechanical department, electronics department, software department and so on). While in the past systems where developed out-of-phase (mechanical design precedes electrical design which in turn precedes software design), nowadays concurrent engineering is applied in order to save development time. However, system-level requirements that cannot be assigned to a single discipline, such as performance, typically cause great problems during the integration phase because the responsibility to meet the requirement is shared among all disciplines. The root cause of this problem is the lack of cross-discipline design interaction. This problem cannot be solved by improving the internal organisation; the way (embedded) software is currently being developed is fundamentally different from, for example, mechanical and electrical design. These engineers basically speak a different language, are concerned about different types of problems and use different techniques to address and solve these problems. This challenge is dominant in the embedded systems domain because the computer and the device it controls both lose their function if they were to be separated. Hence, they cannot be designed in isolation which makes the cross-discipline communication mandatory.

Dependability and fault tolerance. As embedded systems become more pervasive, they face ever more demanding and interdependent functional and non-functional requirements, including the need for reliability and fault tolerance, performance and interoperability. In addition, they are increasingly distributed in character, with multiple processors connected with different forms of network, introducing a wider range of alternative architectures and faults for controllers. In order to achieve dependability, the next generation of embedded systems applications must be engineered to provide predictable behaviour in the face of faults, including malfunctioning infrastructure, environmental hazards, malicious intentions, design defects and degraded component services, any of which can have very damaging consequences if fault tolerance mechanisms are not in place.

1.4 Embedded Systems Design: An Illustrative Story

To illustrate how some of the problems identified in the previous sections can affect a company producing embedded systems under strong market pressures, we present the story of such a development in Fig. 1.2. In this story, a fictional agricultural vehicle company attempts to adapt to a rapidly changing market. Their aim is to maintain their market share by producing a new version of their flagship tractor system, the T1X, while competing against a rival firm. Unfortunately, due to problems during the development, production is delayed significantly and this allows their rival to release a competing product sooner and gain the upper hand in the market.

Although this story is fictional, we have drawn on the experiences of the industrial partners who participated in the development of our method. In the following sections, we consider the different perspectives of the engineers and software designers in this failed development and from this motivate the need for collaborative development that underpins our approach.

1.4.1 The Control Engineers' Perspective

When given the task of building an autonomous GPS-controlled tractor, the control engineers at OPECorp (Fig. 1.2) focus on designing the control laws. They need to find suitable parameters that can cope with the range of different agricultural machines hauled by the tractor and they cannot rely on farmers updating the

OPECorp (Old Physics Engineering Corporation) is a long-established manufacturer of agricultural equipment. In particular they have a historical reputation for producing reliable tractors. While they used to hire mainly mechanical and electrical engineers, in more recent years with the addition of computer controllers to their tractors, they have established a strong software team.

While OPECorp was the market leader for many years, their market share has been eroded recently by competitors producing rival products, in particular Upstart Farm Technologies (UFT). Both OPECorp and UFT produce a range of agricultural machinery (e.g. plows, harrows, seeders) that can be hauled by their tractors. Their tractors provide both hydraulic power and digital data connections for more advanced accessories that sort of follow a standard. The management of OPECorp are keen to maintain their market position and aim to produce a new version of their flagship tractor, dubbed the T1X. The T1X will include a new GPS-controlled autonomous mode. OPECorp expect that UFT are also working on a new tractor on a similar time scale (18 months).

To begin, the control engineers design control laws for the new tractor, which must support the range of agricultural machines that can be hauled by the tractor. Once the control engineers are happy with their low-level control laws, the software team takes over the task of building the software of the T1X. Meanwhile, a GPS unit has been selected and work begins on fitting it to the T1 test rig that the company always uses for new products. With six months to production, UFT announce at an agricultural trade fair that their rival product will contain the latest GPS unit from market leader Loc8, with much higher accuracy than OPECorp's current choice for the T1X. In a surprise move, they also announce compatibility with third-party agricultural machinery from the Ploughman Group (PG). The management at OPECorp insist that these features are incorporated into the T1X, in order to keep up with their rivals. This means the engineers have to fit the Loc8 GPS unit to the prototype and discover that it has to be placed in a different position on the vehicle to the previous GPS unit. They also have to build an adapter for the digital interface used by the PG machines. The software team must now adapt their communication protocols to deal with the PG machines as well. Here it also turns out that the "standard" is not interpreted in the same way by all vendors so it is not as easy as expected.

Once the prototype reaches field trials, the deadlines for the start of full production are looming. The software is loaded onto the prototype machine and trials begin. Unfortunately things go wrong from the start. The tractor drifts steadily from its intended path in autonomous mode. When a PG machine is attached to the T1X, the prototype shuts down, going into safety mode. The trials have to be abandoned and a large amount of money and resources are wasted. It is discovered back at the factory that the new position of the Loc8 GPS unit was not communicated to the software team, which threw off the controllers calculations and caused the drift. The shut down when the PG machine was connected occurred because the software used a library to encode and decode packets used in OPECorp's proprietary communication protocol, which made assumptions about the packets that did not hold true for the PG system. So while the high-level messages were right for the PG machine, the library couldn't decode the low-level packets and didn't recognise messages from the PG machine, causing a safe-mode shutdown. It takes three months to fix these problems found and another month to set up a second set of trials, delaying the release of the T1X significantly and giving an advantage to UFT, whose tractor is released on-time to much praise.

Fig. 1.2 A fictional story of a failed embedded systems development

software once the machine is in production. The different machines hauled by the tractor will affect the balance of the system as a whole, and they want to be sure that the system can be controlled safely and stably. This low-level control is their main area of expertise. To do this, the engineers build a Continuous-Time (CT) model of the T1X and run simulations to tune control parameters that suit the various types of machines.

1.4.2 The Software Designers' Perspective

The software engineers at OPECorp (Fig. 1.2) receive the low-level control parameters from the control engineers, but from experience they know this only forms 10 % of their software for a tractor controller. They have experience in building controllers for the T1 family of tractors and their main focus is on supervisory behaviour. When tasked with adding autonomous control features to the T1X, their experience tells them that mode changes (e.g. changing to and from autonomous mode) and fault tolerance (e.g. dealing with problems caused by the environment that the T1X will find itself in) will form the majority of the control software. To handle this complexity, they build Discrete-Event (DE) models to capture and analyse this mode-changing behaviour.

1.4.3 The Case for Collaborative Development

Heemels and Muller [42] identified a key set of issues that seem to be root causes behind the problems outlined in previous sections:

1. Reasoning about system-level properties is difficult because a common language is lacking. Each engineering discipline uses its own method, vocabulary and style of reporting. This incompatibility causes confusion often leading to misunderstandings and wrong assumptions being made on the sub-designs of other disciplines. These inconsistencies are hard to spot because there is usually no structured system design reasoning process in place.
2. Many design choices are made implicitly, usually based on previous experience, intuition or even assumptions. System-level reasoning is made difficult if the rationale behind such a decision is not quantified. The reasons are sometimes kept hidden on purpose, for example, if strong personal preference or politics plays a role. This may perhaps lead to a local optimum in the system design but only rarely to a global optimum. It is therefore necessary to make design knowledge explicit in order to enable the dialogue at the system level.
3. Dynamic or time-dependent aspects of a system are complex to grasp, and moreover, there are not many methods and tools available to support reasoning about time varying aspects in design, in contrast to static or steady-state aspects.

The effects of these points are amplified by the complexity of the product under development (a typical high-tech system consists of tens of thousands of components and millions lines of code) and the complexity of the design process (number of people involved, organisational structure, out-of-phase or multi-site development, etc.) Our hypothesis is that lightweight and abstract models that capture the system-level behaviour and a reasoning method that indicates how and when to use them will reduce these tensions considerably. A good system engineering methodology will expose implicit or hidden design choices and replace “hand-waving” by design rationale based on objective, quantified and verifiable information.

1.5 A Solution: The Crescendo Approach

The aim of our approach is to bridge the gap between control engineers and software designers by providing tools and methods that allow them to collaborate in the design of distributed embedded control systems. Our approach allows them to explore and test their designs without resorting to expensive prototypes, thereby helping to manage the risks of embedded systems development. The essence of our approach is to use models in all phases of the design. These models describe the system in a more coarse way in the beginning and are gradually refined and extended in later phases. Each result of a step in the design cycle is a combination of design decisions and models, which are used for that decision, and are also the starting point for the next design step. We refer to this as *model-based design*.

A single modelling formalism may be effective for a simple non-distributed system, but it rarely scales. Continuous-time models are excellent for modelling physical system dynamics, but if a distributed architecture is needed or if the idealised assumptions do not hold, the model can become hard to maintain. Discrete-event models are excellent for expressing system logic, making it possible to incorporate a distributed system architecture and failure assumptions, but physical laws are difficult to model. Combining these two worlds, if done in a semantically sound way, can provide a collaborative modelling (*co-modelling*) framework in which it is possible to experiment with both discrete and continuous aspects of the whole system. This makes it possible to run a combined simulation (a *co-simulation*) of both continuous and discrete models, observing the consequences of, say, a change in process distribution or a sensor failure, on the full control system and the controlled process. Changes in models can be caused by changes in requirements or component capabilities, or by deliberate refactoring. In particular, changes in one model may have ramifications on others. The co-modelling approach that we discuss in this book could help address this because the impact of changes can be assessed immediately, prior to the integration of separately developed constituent parts.

To deal with the first five design challenges mentioned in Sect. 1.3, use of models and simulationa needs to start from the beginning of the design process. The Crescendo methods and tooling support collaborative modelling and simulation in all stages of design, including early stages, and so aim to support a concurrent engineering approach. To address the sixth challenge, relating to dependability and fault tolerance, we provide guidelines and examples to enhance initial co-models with realistic behaviour, faults and fault-tolerance solutions. Testing via co-simulation allows designers to explore several possible designs, allowing them to assess these trade-offs in the early phases of the design cycle. As such testing can be laborious, especially when the number of alternatives in combination with several fault situations can explode, the Crescendo automated testing tool can facilitate this work.

But perhaps the most important aspect of Crescendo is not the potential productivity improvement, but the ability to create abstract and multi-disciplinary system-level models that are *competent*. In other words, the predictions obtained from the models match, within some acceptable error margin, the behaviour of the real system that is under development. At the end of the day, this is what modelling is all about, since the overall aim is to raise the confidence in designs, especially in the early phases of the product development life-cycle.

1.6 Conclusion

There is a pressing need for collaborative approaches to modelling in embedded systems design, and presented our approach embodied in the Crescendo tools. In this chapter, we have reviewed the design challenges facing the embedded systems sector, including increased "TTM" pressures and the need for greater dependability. We identified critical problems that occur during developments, including the gap between existing design methods and rapidly changing technology, the challenge of volatile requirements and a need for late closure of the design loop, as well as the necessity for multiple disciplines to collaborate for a design to be successful.

Using a fictional account of an embedded systems design, we illustrated some of these challenges and presented the rather different perspectives of engineers and software designers. Although fictional, the story reflects the experiences of industrial practitioners who participated in the development of the methods and tools described in the remainder of this book. We explained the essential contribution of the Crescendo approach, stressing three key points:

– using a *model-based* approach from the beginning of design;
– allowing multidisciplinary, collaborative modelling; and
– supporting dependability from early in product development by fault modelling, fault injection and automated testing to check fault tolerance.