Preface

Question 1 and 2 of exercise sheet 2 are *not* part of this paper. They were already handed in earlier (question 1) or even voluntary (question 2). However, I will present my solutions for question 3 and 4 here in detail.

Question 3

Describe the relationship between the CMMI process areas Product Integration, Verification and Validation. What does this mean for your day-to-day work as a developer?

By defining the terms "Product Integration", "Verification" and "Validation" we might gain many insights and thus find out how they overlap, it means, what areas they share.

Product Integration

The main goal of product integration is to compose several components in order to obtain a full-featured, properly working and deliverable product compatible to its final environment. Usually, the process works iteratively in a bottom-up manner. For example, the bottom level may be the plain source code consisting of classes while the next one, called architecture components, comprises complex clients/servers, data bases, networks etc. Last but not least, integrating the product in its final environment completes the process. The "human factor" of product integration such as training, documentation, deployment etc. needs to be mentioned, too, because it consumes much of the time and therefore yields lots of the entire costs.

Common interfaces (technical and the human ones !) are the basic elements required for successful product integration. They should be defined as early as possible, even if they fulfill just internal tasks. Sometimes these interfaces are thought of as a contract that clearly states how the flow of information and commands must be handled, foremost their syntax and their semantics.

Interfaces can be defined (and will be found) at any level of the iterative process. In order to properly evaluate a level's compatibility, one has to ensure that there are no open issues at any lower level. After all, one will determine and generate an integration sequence describing which steps have to be taken in which order to correctly deploy the product as intended.

Verification

The process of verification compares the actual product or parts of it to its/their specification by means of completeness and correctness. The most well-known techniques used for Verifications include (peer) reviews, tests and static analysis.

Validation

Validation evaluates whether the product fulfills its intended use when placed in its intended environment. For sure, validation can't be performed as a pure formal process as it covers the aspect of satisfying the customer who wishes to obtain a product accomplishing his or her requirements.

Relationships

I found a well-designed diagram describing the V-model in the most comprehensive German book on software engineering (/Balzert 98/). It incorporates validation and verification in the software development process quite handy. I added product integration to emphasize its importance and to clarify its significance. Despite the fact that I don't know the correct English translations of terms used in that figure, they remain unchanged in German.

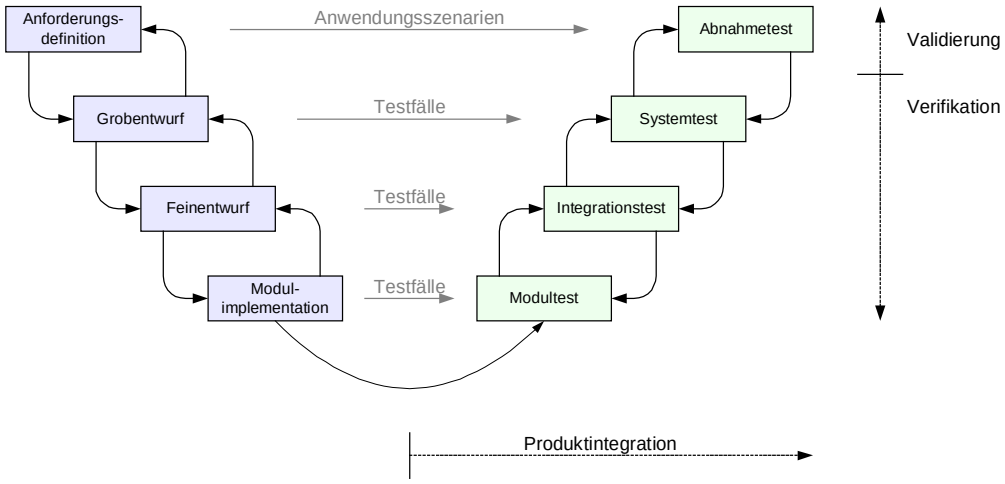

Figure 1: V Model

Verification can and should be done at any level or stage of the V-model process except for the first and last one. Therefore we get some kind of a hierarchy looking much like the bottom-up approach of product integration. Indeed, verification plays an substantial role in product integration: an accurately integrated product has to be verified first to ensure a flawless interoperation with existing external products.

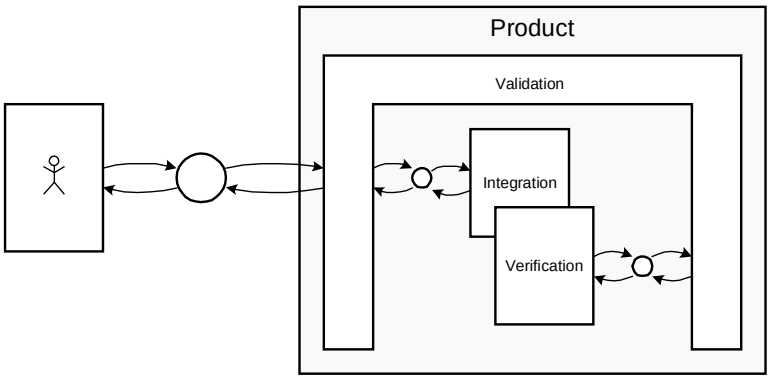

Figure 2: Relationship Block Diagram

The customer doesn't want to worry about technical issues such as verification. His only interest lies in a successfully validated program because that empowers him to actually use it in order to get his special tasks done. A tight and seamless integration into the working environment is part, or better to say: source, of the customer's satisfaction and any result produced by the program must be as accurate as required – not necessarily more.

In my eyes, integration can be traced down to a blend of verification and validation because all you do is to verify all interface if they are correct and validate if they are complete. If that holds true, a huge step is taken.

Most of the software projects I participated in suffered from a weak or even lacking verification strategy. Additionally, product integration was performed less than optimal. Both drawbacks are rooted in a steady demand for cost cutting and milestone shifts forced the middle or upper management in these companies.

According to Figure 1, validation takes place at the end of the development process. Nevertheless, it refers to assumptions and commitments accepted in the very first phase. Thus, requirements management, the initial phase, needs to get more attention – for example by allowing a broader time span.

Right from the beginning a good schedule should set fixed milestones. Herein all developers verify and validate the current prototype in order to detect flaws and errors as early as possible. Especially validation proved to be a very strong and strict tool because often the customer didn't know what he actually wanted. A horizontal prototype usually helps both parties in creating a more detailed specification.

Question 4

For each engineering process area describe typical methods and tools that can be used to implement the activities of that process area. Don't just list the methods and tools but explain what part of the process area is supported by them.

Requirements Management

Interviews build up the foundation of any requirements management since they offer the ability to direct the "investigating" process by both the customer and the software developer. It should be noted that interviews are not restricted to the upper levels of the companies, instead, one has to ask the potential end-users who often reside on lower levels. The customer's satisfaction relies on contented users.

Depending on the process model, a Vision Paper or a Lastenheft (comprises intended work) contains the results of the interviews held in a written, fixed manner. Up to now, I didn't use any supporting software tools in requirements management but I am quite sure that lots of them exist (Rational ClearCase might be one of them).

Requirements Development

The Pflichtenheft is the very formal counterpart of the Lastenheft and describes all requirements in a contract. Therefore, any participated person – either customer or developer – may sue the other one if that contract is not fulfilled as promised.

RAD (Rapid Application Development) helps to set up either horizontal or vertical prototypes. It turns out that customers can easier detect missing features when "playing" with the prototype since thus their creativity is supported. The prototype ought to include the major use cases of course.

There are many techniques to estimate the following efforts: time, staff and money. The latter falls into the scope of economics – I omit it since each mid-sized company employs some specialized persons caring about it. Maybe they run SAP R/3 or something similar. Time and staff can be efficiently managed with Microsoft Project (and SAP R/3, etc.). These programs evaluate the plans, discover overlaps and estimate resource usage. Common techniques, such as Gantt diagrams or the Function Point method, are originated in the area of economics, too.

Rational Unified Process might help but unfortunately the HPI owns no licence yet. So I hadn't the chance to verify anything I read on the internet. Colleagues told me that Merant PVCS is comfortable, too.

Technical Solution

The first step in building a huge application is to model its structure, interfaces and data flows. Recent methods like UML cover the field of object-oriented techniques (OOA, OOD). Older approaches (SA, SD) should at least be extended by the large-scale modeling idea of FMC (outcome of intense research at the HPI). A decent integrated development environment comes along with modeling support. Examples include Borland JBuilder and Microsoft Visual Studio .NET or third party add-ons like Rational's suite.

The actual process of writing code – based on the modeling concepts – is subject to be revolutionized in the next few years. It has been shown that the quality and reliability of code generators is steadily increasing so RAD and especially component-based approaches massively gain ground when it comes to build GUI based applications. Even the way back, to re-model when the team discovered design flaws and had to work around them, seems to be working at a fair level.

However, coding is about social behavior. Most of the failed software projects can be traced down to human factors. Therefore, during my bachelor's project we followed the emerging trend of eXtreme Programming. It worked out fine for the reason that our team has been very small: just three persons. For bigger teams, other role-based teamwork models have to be analyzed.

As the number of developers rises, the urgent need for a decent and sophisticating documentation plays an important role. There a two basic varieties: code-based and external. The first one can be done via comments in the source code whereas the second utilizes a combination of text and graphics. Two good examples are Doxygen (code-based) and Microsoft Office (external; incl. Visio).

The safe and reliable distribution of code among many developers or even teams is the business of CVS programs. In addition, groupware solutions provide a commonly shared communication platform.

So far, I don't have any experiences concerning the tools Together or Innovator. They seem to be full-featured suites empowering the developer to finish his tasks in the shortest time possible, too.

Product Integration

The market of setup generating tools has been grown in the last years. One of the most seen program is InstallShield whereas Microsoft is forcing a new shift in the market: they built in a entirely free but powerful API in their operating systems. Additionally, Visual Studio .NET brings along a MSI package generator that takes care of dependencies, older versions, repair mechanisms, DLLs, etc.

A further new paradigm are web installers. They download the most current version and exclude all unnecessary parts; Microsoft's installers are capable of that, too. A bit older but still important today is network-wide deployment, unfortunately an area where I don't have any own experiences yet. New middleware technologies such as the Microsoft .NET framework support install-by-copy. The API allows to inspect available components at runtime and configure them appropriately.

Today you still will find the widespread use of conventional distribution schemes like CDs. Many burning tools (Nero, etc.) offer a great variety of options to enhance installations or maintenance by creating bootable discs or portable file systems. These storage mediums or the stored files in turn can be compressed (Zip, SFX, Tar/GZ).

In my opinion, product integration is *the* area where Unix based systems lack a lot when it comes to software development. Incredible long makefiles and init-scripts are not needed on the Windows platform. After all, Windows' abstraction layers hide the actual hardware behind some unified API calls and thus reduce dependencies. One example is the printer spooler for superior to CUPS or similar Linux/Unix complements.

Verification

A first step of verification is debugging. The IDE *must* support it in order to be called a true IDE. Nevertheless, some debuggers for specialized purposes exist: SoftICE or other into-the-kernel debuggers go even one step further than the common debuggers do.

Again, Rational provides advanced tools like Purify that detects common mistakes (like lint) and memory leaks. The very formal analytical tests like attempts to analytically prove correctness remain extremely expensive, hence I never saw a tool supporting it. For ultimate security, there are in utilized in companies or organizations like NASA, Boeing or Raytheon.

Some say that syntax checks of compilers are a kind of verification, too. I have to agree since I run many scripts on private servers that were not properly checked in advance (they are just interpreted) and suddenly terminate because of syntax failures what I find very annoying.

All major applications include a test framework (JUnit etc.) to output log files. In case of a crash, conditions leading to the program's termination are either displayed by the program itself or the hosting operating system (well-known blue screen). The core dump or memory dump should help experienced programmers to track down the problem's source.

The key idea of XP, abbreviation of eXtreme Programming, to review in small teams consisting often of only two persons, has been to me the most efficient method to *prevent* bugs. Tricky and nasty bugs deep inside the program structure can be found afterwards by performing peer reviews or walkthroughs as well.

Modern application work across system boundaries. Therefore, it is the developer's responsibility to guarantee portability and interoperability. Nowadays, middleware like CORBA or the .NET framework takes care of these issues. To verify components independently, test driver, dummies and stubs emulate an environment.

Validation

As already explained in this paper, prototypes offer an early opportunity to validate whether the product reaches the required level of completeness, quality and performance. These sample installations undergo several stages: alpha test (some functionality may be missing; usually on the developer's machines) and beta test (all features implemented but verification not finished; usually on selected customer's machines).

The degree of completeness, quality and performance can be measured. The established metrics include: source-level coupling, errors per line of code, and throughput per second.

Especially the system's performance under heavy load may be interesting. Even a perfectly verified application containing not a single error should show a well-defined behavior, e.g. a reboot may be acceptable to avoid the crash of connected hardware and/or software.