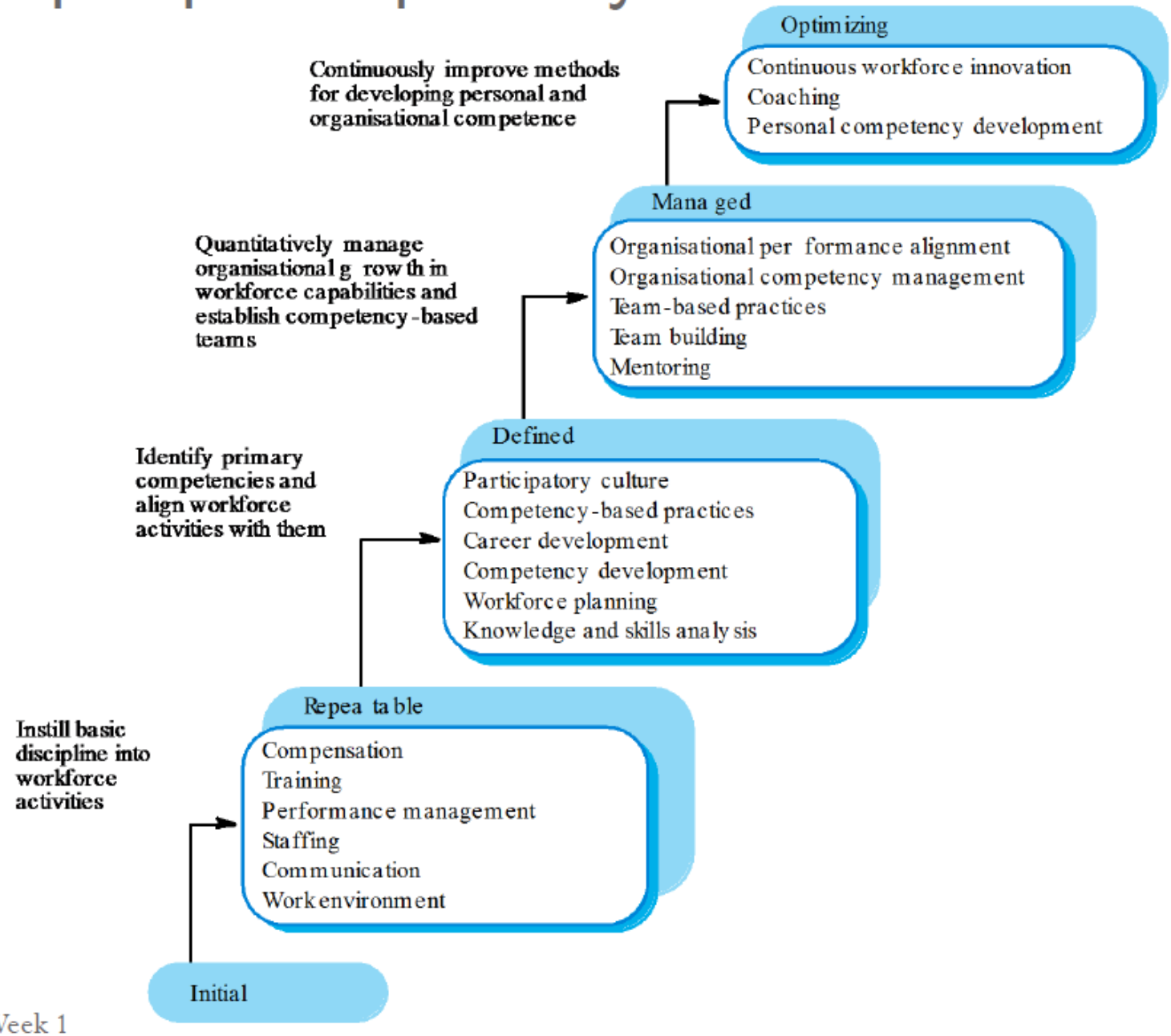
SQE in Agile Environments

- CMM: Greater control over the whole process
- CMM: More about paperwork and management than real work
- Agile: Constant dialogue between “possible” and “desirable”
- Agile: Small releases, simple designs, constant refactoring
- Agile: Pair programming, content ownership, continuous integration
- Agile: Advantages/Disadvantages

The people capability model



Five stage model

- Initial: Ad-hoc people management
- Repeatable: Policies developed for capability improvement
- Defined: Standardised people management across the organisation
- Managed: Quantitative goals for people management in place
- Optimized: Continuous focus on improving individual competence and workforce motivatio.

CMM versus Agile

- CMM and Quality: Many ideas behind CMM are taken directly from statistical product quality work, particularly the principal idea that organizations improve quality by gaining increasingly **greater** (statistical) **control** of their processes.
- The idea of capability levels actually comes from the **quality world**; indeed, both the terms and definitions for the levels (initial, repeatable, defined, managed, and optimized) are taken directly from other quality work.
- Becoming a CMM Level 3 organization has become a widely accepted certification in software marketing. Becoming a Level 4 or 5 organization is supposed to make you a top software engineering firm.

CMM levels

Level 1: Initial

- Level 1 is fundamentally a software process that is ad hoc. There is no specific methodology, and each project is more or less a new activity. The initial stage is the default in software management. According to Joseph Raynus, principal at ShareDynamics, "Unfortunately, the initial level processes are the most practiced processes in the software business".
- Ultimately, the most serious problem with most Level 1 organizations is that management doesn't realize it's at that level.

Level 2: Repeatable

- Basic management exists within Level 2 organizations. Level 2 organizations have a **basic project management structure in place that tracks costs, schedules, and, to some degree, systems functions**. As the name implies, the software process in place is repeatable and reinforced by training and management controls.
- Repeatable level organizations **use the same processes time after time**. Because the organization has repeatable processes, it can begin to work on measuring and improving these processes. It can also begin to work on the development process itself.

CMM levels

Level 3: Defined

- At Level 3, a software development organization has a well-defined software engineering approach as well as a project management framework. Here, the processes are well documented and communicated across the organization through standards and training. Raynus says, "If the repeatable level defines what to do and who should do it, the defined level specifies when to do it and how to do it".
- Though Level 2 organizations are mostly concerned with project management processes, Level 3 organizations are, theoretically at least, also concerned with **defining and measuring software development processes**. This difference represents a major difference in quality.

Level 4: Managed

- At Level 4, software development is increasingly **managed using statistical methods**. Both process and product quality measures are collected and monitored to quality controls. At Level 4, software products are not only managed to meet time and cost schedules, they are also managed to meet specific customer satisfaction and defect levels. Level 4 organizations understand and practice statistical quality control.

CMM levels

Level 5: Optimizing

- At Level 5, organizations **reach quality heights**. Software is produced as in Level 4, but the organization is continually working to improve its processes and standards. Level 5 organizations are **world-class innovators in software development**.
- The idea behind CMM is that through a certification process, organizations can be checked out, their current state can be identified, and management can be provided with a roadmap for improving their performance. By committing to the CMM process, organizations can start wherever they happen to be, and through a conscious statistical quality control program can begin to improve. All it takes is commitment, training, measurement, and correction.
- In a recent conversation with Council Fellow Ed Yourdon, he discussed the experiences of the Indian software company whose board of directors he sits on. This firm had embarked, as had so many Indian firms, on **CMM to help in marketing to the US**. Over time, however, they had become such experts in its use that they are now using it on smaller and smaller projects. Indeed, it is now part of the organizational culture. This organization had pushed even further by putting management practices on the Web for all of their clients to see in real time.

CMM advantages

- In Australia there are **zero companies** at level 5 and just 2 at level 4 (Motorola is one of these)
- Like many quality-based initiatives, CMM appeals to organizations and managers interested in **control and certification** since it gives them some objective qualification of their software development process. The process is normally that an outside CMM organization will come in to certify an organization for a specific CMM level. **This is important in organizations where CIOs and others are looking for ways to add outside creditability**.
- CMM certification was first required, as you might expect, on military projects, and, over time, it has been extended to a variety of federal, state, and local agencies. Today, many large private organizations lean heavily on CMM certification as a means of screening software vendors. As previously mentioned, certain outsourcing vendors see high-level CMM certification as both a marketing and management tool. By showing that they are independently certified, they increase their creditability and reduce client concerns about moving development out of house or out of the country. Clients can track in real time on time and cost processes.
- CMM is clearly the best-known, best-documented software management method in the world. Being recognized as an industry best practice has many advantages to an organization trying to improve itself.

Concerns with CMM

- On the negative side, many people have argued that **CMM certification appears to be somewhat arbitrary**. Organizations that certify in CMM make their money by showing that it has helped organizations move up the ladder to Level 5. Getting certification, at least at Levels 2 and 3, often seems to be more associated with **good paperwork than with good management**. It is, after all, very difficult for an outside organization to come in, and, in a very short time, determine whether an organization is doing what it says it's doing.
- A second major concern is that CMM certification is largely about an organization's **management processes** (estimating, scheduling, control) and **not nearly so much about the quality of the software products produced**.
- There are even those who say that to become a Level 2 or 3 organization, a company doesn't necessarily need to have a good (or state-of-the-art) software process in place but only must blindly follow whatever process it has.
- Other critics say that CMM certification is not so much of a quality certification, but a marketing one. As with ISO 9000, organizations seeking CMM certification bend over backward to **produce reams upon reams of documentation** that can then be reviewed by CMM inspectors.
- No matter what your viewpoint, though, one thing is clear: CMM **Level 3 or 4 ordinarily involves a very strong (heavy-handed) management style**. By and large, more attention in these organizations is paid to project management than to actual production. This is a matter of concern for many software developers who would much rather design and program than fill out project management forms.

Agile

- Systems are best developed in short increments.
- Users and developers must work hand in hand.
- Each systems increment should be designed to handle the minimal requirements.
- When changes in requirements occur, the agile development team believes that they should be designed in rather than added on and that there should be minimal documentation beyond the code.

The Planning Game

- "software development is always an **evolving dialogue between the possible and the desirable**". **Neither the customers (business users) nor the technical folks should have complete control**. The business users decide scope and priorities, as well as the time and scope of the product releases. The technical people, on the other hand, decide estimates, technical consequences, the development process, and detail scheduling. In XP projects, the planning game is a give-and-take process emphasizing **true collaboration**.

Agile

The Agile Manifesto

- Satisfy the Customer through **early and continuous delivery of valuable software**
- **Welcome changing requirements** – harness change for competitive advantage
- Deliver working software **frequently**
- **Business people and developers work together** daily throughout the project
- **Face-to-face conversation** is the most efficient and effective method of conveying information
- **Working software** is the primary measure of progress
- Sustainable development means that sponsors, developers and users should be able to **maintain a constant pace** indefinitely
- **Constant feedback** – stay on track through daily short meetings
- **Customer involvement** using facilitated workshops or focus groups
- **Technical excellence** – using refactoring creates business value now and in the future

Agile

Small Releases

- "Every release should be as small as possible, containing the most valuable business requirements “. A doable project is one that is small enough to be done quickly and big enough to be interesting to the business, customer, etc.

Metaphor

- Every project should have a single, overarching concept or metaphor. Metaphor is intended to replace the term architecture within a project and provide it coherence. "As development proceeds, and the metaphor matures, the whole team will find new inspiration from examining the metaphor".
- Of all the components, the metaphor is perhaps the most difficult one to grasp. At some level, it is supposed to be the essence or the irreducible core of whatever it is your project is building. When done right, it makes it possible for team members to keep focused.

Agile

Simple Design

"The right design for software at any given time is one that:

- Runs all the tests.
- Has no duplicated logic. Be wary of hidden duplication like parallel class hierarchies.
- States every intention important to the programmers.
- Has the fewest possible classes and methods."

“Simple" has lots of meanings to lots of people, and I can imagine a number of scenarios where the definition of simple might not turn out to be so simple after all.

Agile

Testing

- "Top XP teams practice '**test-driven development**,' working in very short cycles of adding a test, then making it work. Almost effortlessly, teams produce code with nearly 100% test coverage, which is a great step forward in most shops,".
- Test-driven development means that agile development **starts by defining tests before developing code**. It is easy to overlook the importance of this idea. In doing so, programmers are forced to work out the inputs and outputs that they are trying to achieve and then to build code that produces the correct results from the predefined tests.
- Like so many factors in agile development, the basic strategy rather than a set of heavy-handed management controls is responsibility for good code and good programming discipline.

Agile

Refactoring!

- Refactoring involves the constant redesign of the entire OO class hierarchy, and methods such as new requirements come to be known. Agile development programs evolve through iteration. At each release, new requirements are introduced that mandate the required design changes.
- Refactoring goes hand in hand with the idea of minimal requirements. One of the basic tenets of XP, and therefore, agile development, is that of implementing only those requirements that are known for certain and that are involved in the current release.
- Historically, one of the problems with radical iterative design is that after a number of iterations, the program(s) becomes increasingly difficult to maintain. As change after change is added to the program, it becomes more and more difficult to understand. However, with the idea of refactoring, the design is revisited at each iteration, and changes are integrated, thus making the systems evolution smoother.
- If agile development's form of iterative development with minimal documentation is to succeed, then there must be an increased emphasis on design. Refactoring really means design/redesign.
- Early versions of OO design emphasized spending enormous amounts of time designing class hierarchies and methods before building an application using the classes and methods. But this made early OO very front-end loaded. A great deal of time was usually spent on the initial classes, only to discover that you always had to redo them as the project progressed and you learned more about the problem. In OO, this became known as "class thrashing."

Agile - refactoring

- In agile development, there is minimal design up front, but that minimal design is repeated for each iteration. The belief is that if you do it right, the program will improve, not degrade over time. There are some difficulties with this approach, however. For one, **manually refactoring OO code is not an easy task**, and, for another, some kinds of refactoring (i.e., data refactoring) are much more difficult than others.
- Even though agile development, coming as it does from OO, is a highly technical approach, there is discussion about **the importance of automated tools** in such critical areas as class and method refactoring. Because of the property of inheritance, changes in one class often ripple through a system in unpredictable ways. Recently, **tools have begun to make this process faster and more reliable**.
- Class and method refactoring are only one part of the problem. An equally large problem that has been largely ignored by agile development is the importance of data refactoring. Most of the major problems associated with redesign occur whenever it is necessary, for whatever reason, to **redesign the underlying database**. Historically, OO design has been somewhat light on database design and redesign. It is important to have data refactoring and automated refactoring tools, of which automated database redesign tools are the most important.

Agile

Pair Programming

- All programming is done by pairs of programmers working so closely together that one can pick up and/or modify the work of the other at any time.
- Agile developers maintain that pair programmers working together are far more productive over the long haul than two programmers working independently. They maintain that by working in tandem, the design of each part of the code is made more clear and easier to comprehend and that, by working in pairs, individuals learn better habits and produce more than they would on their own.
- The pair programming approach institutionalizes the practice of code reviews. Each day, each pair programmer must understand all of the code that he or she is responsible for no matter which partner writes it. Agile development maintains that the result is better, producing tighter code that is easier to understand. Used in conjunction with test-first development, pair programming speeds the development process by forcing both programmers to use the same (or at least a common) design strategy.
- Pair programming in an agile development or XP environment creates an atmosphere where people know more because they are involved more. According to Beck, "If two people pair in the morning, in the afternoon they might easily be paired with other folks. If you have responsibility for a task that is unfamiliar to you, you might ask someone with recent experience to pair with you. More often, anyone on the team will do as a partner" .
- Pair programming is knowledge management par excellence. By making sure that for every function there are two people who completely understand what is going on, there is far less reliance on one key individual who might leave, get sick, or otherwise become unavailable.

Agile

Collective Ownership

- "In XP, everybody takes responsibility for the whole system,“. In agile development, the central product of the organization -- the software -- is owned by the group. Instead of individual programs being more or less exclusive property of one individual from concept through turnover to maintenance, the entire set of code is the property of everyone in the group. Anyone can look at any piece of code and can, within the boundaries of pair programming, change anything as well.

Continuous Integration

- Because of its emphasis on speed, agile development focuses on constant integration. Rather than have a set of periodic program (system) builds, the code base is in a state of continuous integration. New programs can be added at any time. The pair programming teams integrates the code base. If something fails, typically the team that has made the most recent changes is normally the group that tracks down the problems. It is well known that the shorter the cycle between coding and testing, the better the ultimate end product.

Agile

The 40-Hour Week

- Agile developers state that software development is more like a marathon than a sprint. In knowledge-based work, it is important that the individuals that make up the team be as fresh as possible. Although there is not much that software development managers can do to help team members avoid stress and burnout at home, they can certainly do so on the job.
- The rule, then, on agile development projects, is that people in the normal development cycle only work 40 hours a week. Though there may be occasions for overtime, there shouldn't be overtime worked more than one week at a time: in other words --no death march projects.

The On-Site Customer

- States Beck: "A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. By 'real customer,' I mean someone who will really use the system when it is in production" .
- This is absolutely true of the customer/user on an agile development project. People who are going to be using the system are responsible for defining how that system is going to work, and their commitment is for the period of the entire project.
- The total commitment of the user is one of the major hallmarks of an agile development project. Users are as responsible for the definition, user interface, review, test case presentation, and prototyping of the end product as anyone on the team. Because of this, there is very little finger-pointing when the product is delivered for broad installation and use.

Agile

Coding Standards

- Finally, there is the issue of coding standards. Because ideas of pair programming, collective ownership, and minimal documentation, it is critical that **all of the code follow very strict guidelines**.
- In this environment, "you simply can't afford to have different coding practices. With a little practice, it should be impossible to say who on the team wrote that code".

Agile

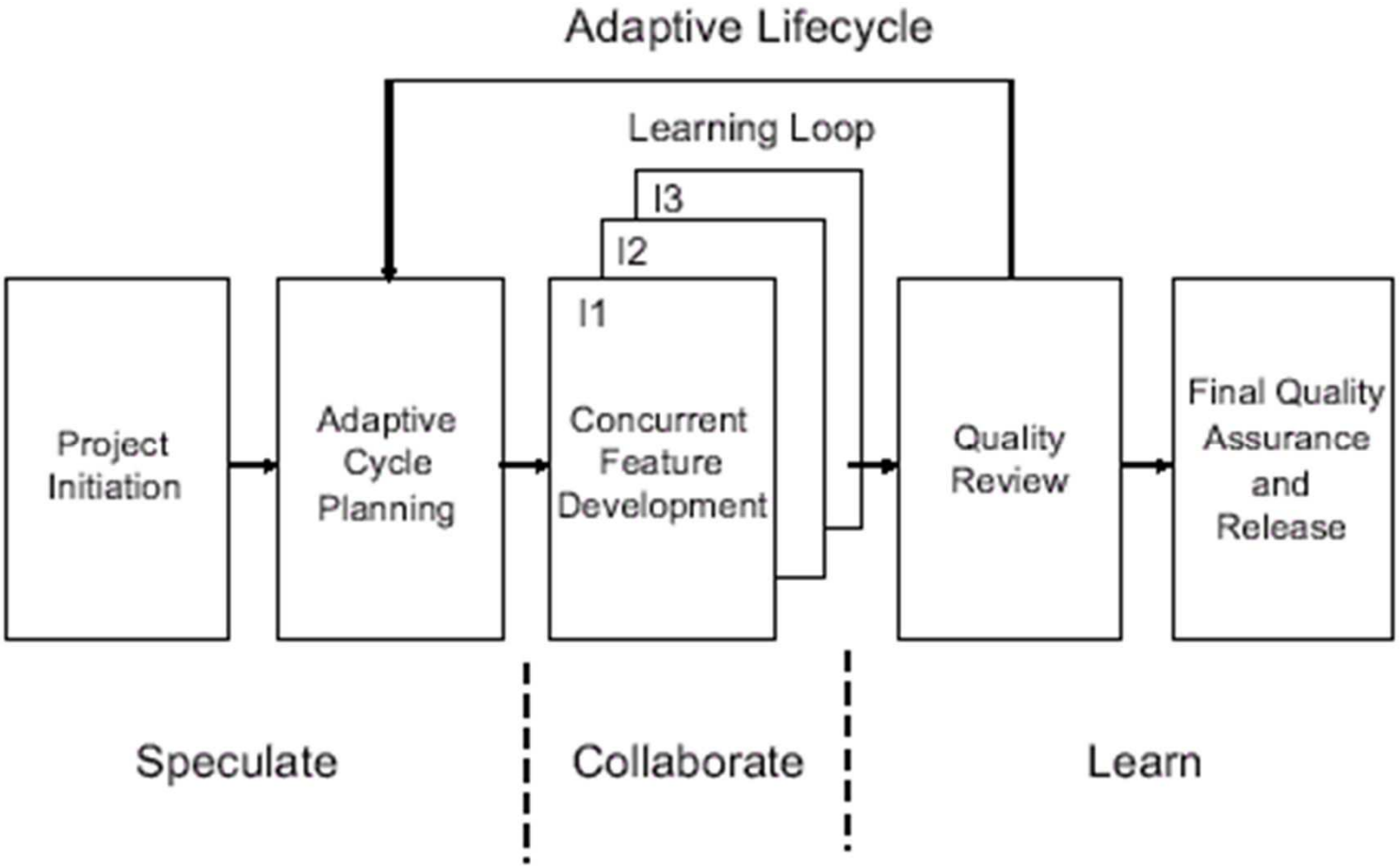
The Management view

- Slash the budget – small budgets force developers to focus on the essential
- If it doesn't work, kill it – triage – is it doing what the company needs? Is it working?
- Keep requirements to a minimum – start with what the software MUST do, only 7% of features are actually needed

Agile

- Build on success, not failure – as often as once a week, complete a piece of software and have your business deciders test and approve it
- Keep the development team small – team coding is more efficient and produces stronger code – Caterpillar's Agile team of 15 over 2 years produced 200 screens and 1.5 million lines
- Assign non-IT executives to software projects – the perfect developer would be half businessperson and half programmer

Agile Lifecycle



Agile: Some additional aspects

- Agile development simply must produce more design artifacts. The idea that the code ought to be the only documentation you'll ever need doesn't cut it.
- Agile development has to get over its aversion to relying on tools. The software business has made itself one of the key drivers in modern business by providing electronic tools.
- Today's business world is an electronic one that operates in real time. Software development has to keep up. No matter how good our management strategy for software development, we can't get to where we have to without better and better tools.