




Best practice and design pattern for modern web application building: an architectural view

NOTICE OF CONFIDENTIALITY
This document contains information that is confidential and the property of base16 srl. This document and/or its contents may not be revealed or disclosed to persons outside the entity it was prepared for, nor may it be used, copied, reproduced, modified, transmitted or republished without the prior express written permission of base16.



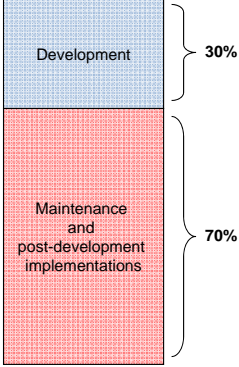
> SW developers 2

Pursuing a career as a Software Engineer

- Anyone interested?
- Thoughts, ideas, how do you think it works?
- What is software development about?
- Where, who, when, how much and why.
- How does the market look like?

<base16/>
THINKING HEX
> The big challenge
3

Enterprise software TCO structure (5yrs basis)



Category	Percentage
Development	30%
Maintenance and post-development implementations	70%

“(...)Enterprise software has gained a well-deserved reputation for lengthy and expensive implementations, followed by high ongoing costs and inflexibility(…)”

John F. Martin, Senior Vice President, Corporate & Product Strategy for IQNavigator, Inc.

- > Gartner has estimated that up to 70% of TCO is due to post-implementation modifications;
- > Only 10% to 30% of coding time is used to develop value added services and core functionalities;
- > Enterprise software usually requires large upfront investments and a huge amount of trust;
- > Ongoing benefits tends to zero (and less), due to increasingly maintenance, administration and update costs (again TCO);
- > Business moves faster than software developers.


<base16/>
THINKING HEX
> The big challenge
4

Any idea?

<base16/>
THINKING HEX

> The big challenge 5

A good place to start:



- >Solid design principles and patterns
- >(Self-)Controlled* and quality-oriented processes
- >People and spirit

* Controlled not meaning having someone on-your-back every second, but meaning measuring progresses and gathering feedbacks: always seek for improvements. In other words: you control what you measure.

<base16/>
THINKING HEX

> The big challenge 6

Does it really make sense?

It takes time and money to plan a proper design. It takes more time and more money to enforce a solid process.

How can adding more complexity help contain development costs?

Well, ... do not focus on the short term, but look at the TCO.

<base16/>
THINKING HEX

> SOLID: it's not a casual word 7

- > **Single Responsibility**
If there is more than one reason for a class to require a change, this principle is violated
- > **Open-closed**
A class should be open for extension but closed for modification
- > **Liskov substitution**
objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- > **Interface segregation**
Many very specific interfaces are better than one generic
- > **Dependency inversion (by injection)**
Depend on abstractions

<base16/>
THINKING HEX

> Single Responsibility 8

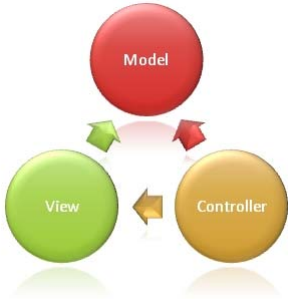
SOLID_SRP.rtf
SOLID_OC.rtf
SOLID_DI.rtf

<base16/> THINKING HEX > Altri pattern e principi 9

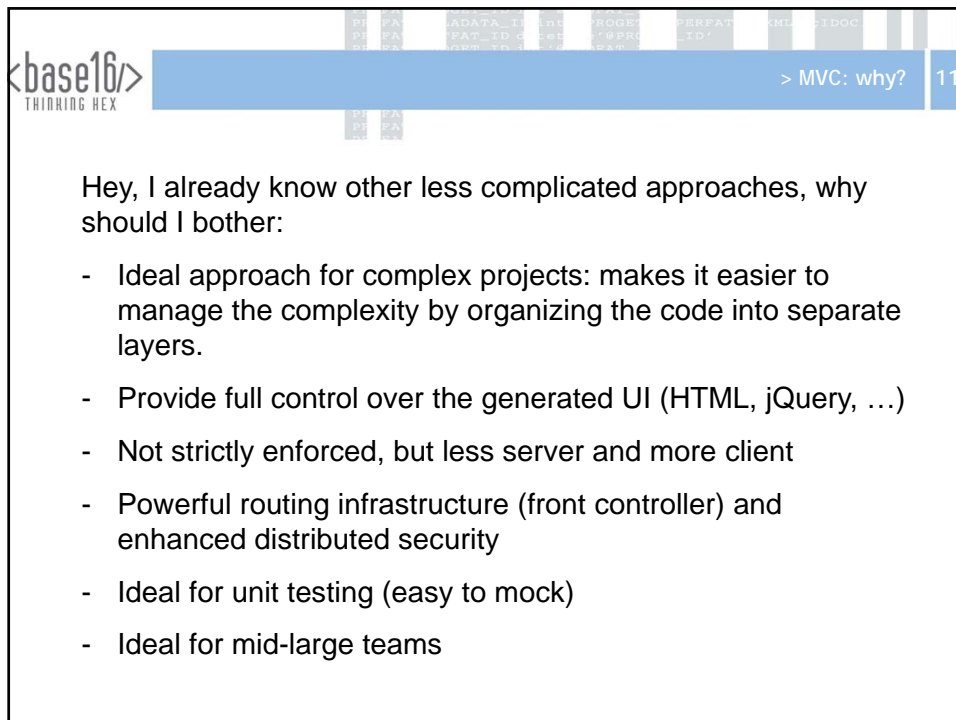
There is much more than SOLID principles:

- Commonly used design patterns (creational, structural, behavioral)
(<http://www.dofactory.com/Patterns/Patterns.aspx>)
- SoC (Separation of Concerns): layered designs;
- MVC (Model-View-Controller): software architecture which enforce SoC.

<base16/> THINKING HEX > MVC 10



- View (UI): Components which represent the UI
- Model: once called “business logic”, represents the objects manipulated by the applications and the related operations
- Controller: Handle the user interactions, interact with the model layer to retrieve/store/manipulate data and push/get data to/from the views



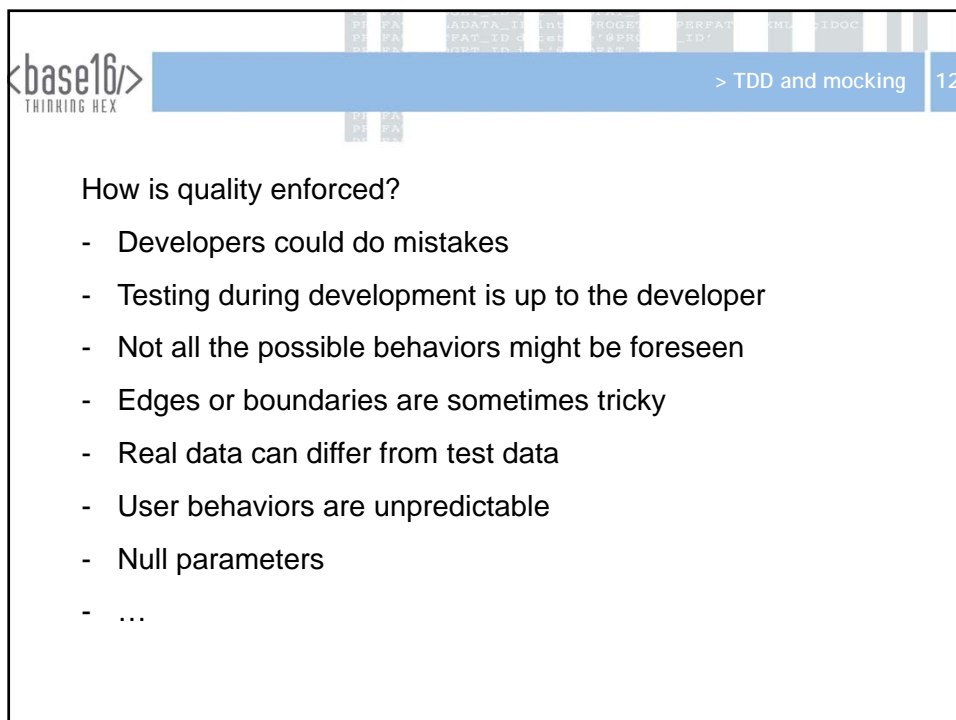
The slide features a header with the Base16 logo on the left and the text '> MVC: why?' on the right, next to a page number '11'. The main content area contains a question and a list of reasons.

<base16/>
THINKING HEX

> MVC: why? 11

Hey, I already know other less complicated approaches, why should I bother:

- Ideal approach for complex projects: makes it easier to manage the complexity by organizing the code into separate layers.
- Provide full control over the generated UI (HTML, jQuery, ...)
- Not strictly enforced, but less server and more client
- Powerful routing infrastructure (front controller) and enhanced distributed security
- Ideal for unit testing (easy to mock)
- Ideal for mid-large teams



The slide features a header with the Base16 logo on the left and the text '> TDD and mocking' on the right, next to a page number '12'. The main content area contains a question and a list of points.

<base16/>
THINKING HEX

> TDD and mocking 12

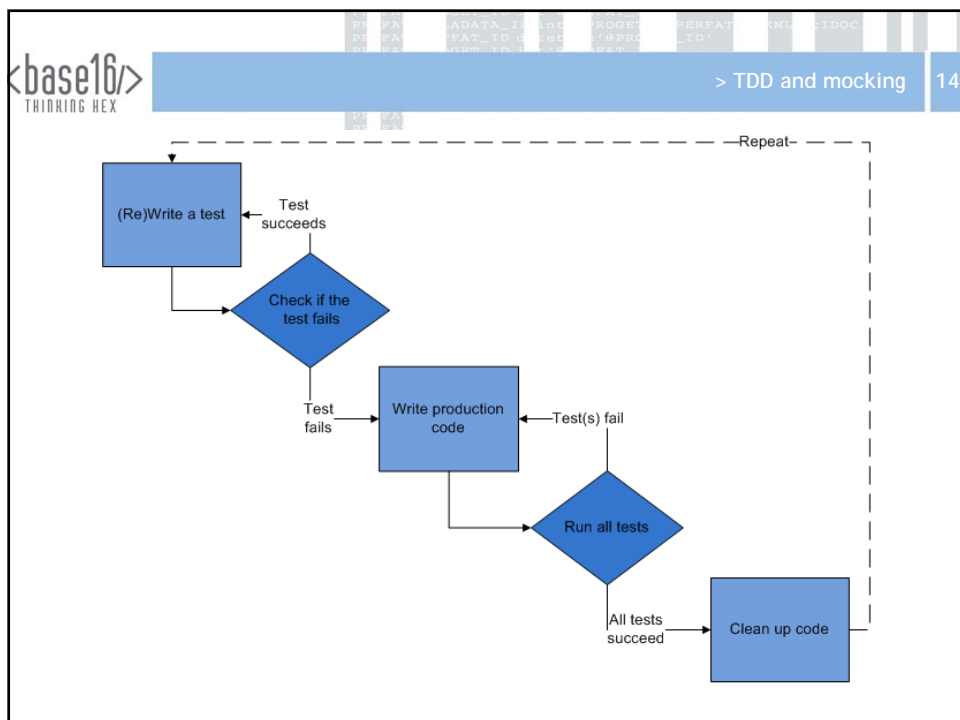
How is quality enforced?

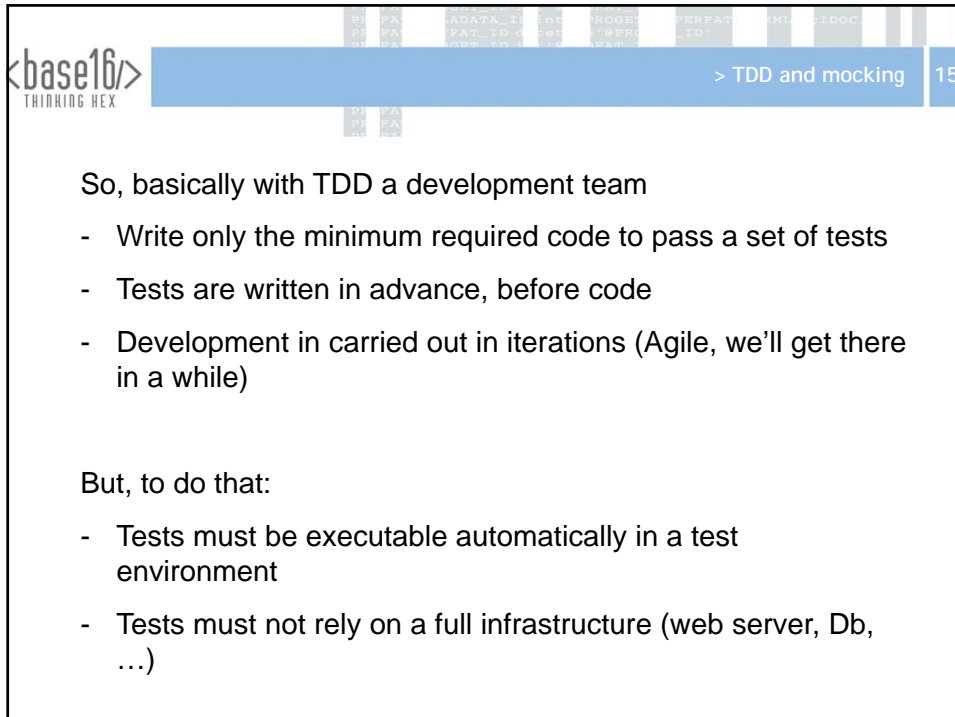
- Developers could do mistakes
- Testing during development is up to the developer
- Not all the possible behaviors might be foreseen
- Edges or boundaries are sometimes tricky
- Real data can differ from test data
- User behaviors are unpredictable
- Null parameters
- ...

<base16/> THINKING HEX > TDD and mocking 13

As seen at the beginning an application costs because of its maintenance, meaning that since developers have to chase bugs around and solve them, it cost a lot of money.

➡ Wouldn't it be nice if at the end of development the application was already tested?





base16/ > TDD and mocking 15

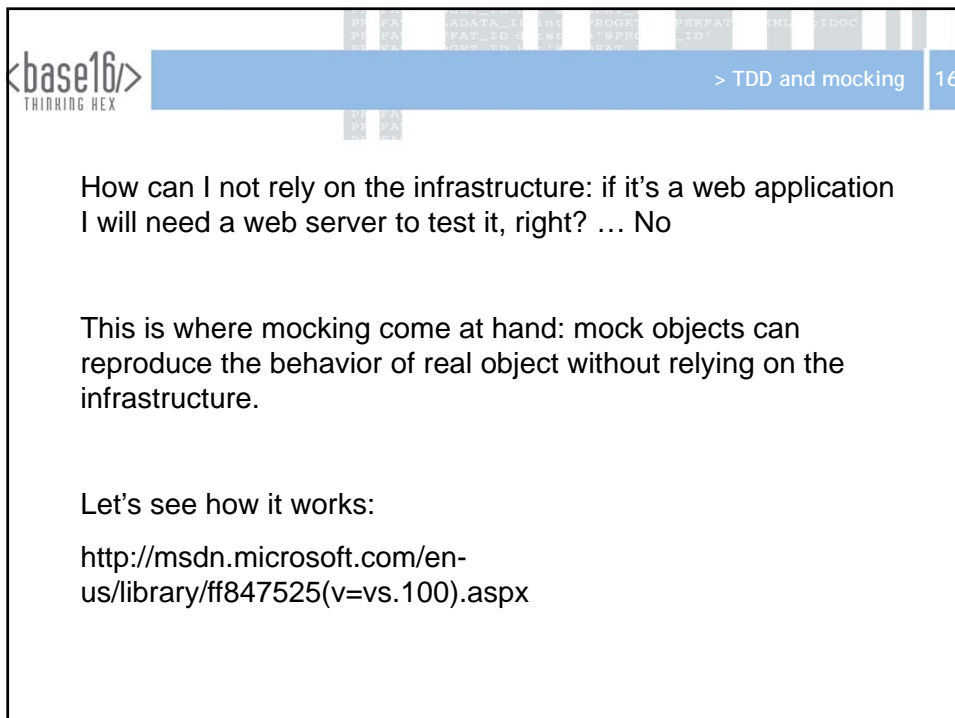
THINKING HEX

So, basically with TDD a development team

- Write only the minimum required code to pass a set of tests
- Tests are written in advance, before code
- Development is carried out in iterations (Agile, we'll get there in a while)

But, to do that:

- Tests must be executable automatically in a test environment
- Tests must not rely on a full infrastructure (web server, Db, ...)



base16/ > TDD and mocking 16

THINKING HEX

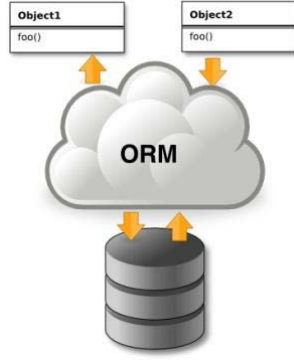
How can I not rely on the infrastructure: if it's a web application I will need a web server to test it, right? ... No

This is where mocking comes in: mock objects can reproduce the behavior of real objects without relying on the infrastructure.

Let's see how it works:

[http://msdn.microsoft.com/en-us/library/ff847525\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/ff847525(v=vs.100).aspx)

<base16/> THINKING HEX > ORM 17



What happened to the data layer?

Modern applications relies on ORMs (Object-Relational Mapping) which expose and persist object states into database tables and vice versa.

A layer of entities is generated (model first or database first) along with their CRUD methods and exposed to the application.

<base16/> THINKING HEX > ORM 18

Most common benefits:

- Abstraction of the database (independence from a specific database object or from a DBMS)
- Design-first approach
- Separation of Concerns
- Optimization (automatic caching, connection pooling, ...)
- Automatic type conversion

<base16/> THINKING HEX > ORM 19

```

namespace MigrationsDemo
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
    }
}
...
Blogs.where(x => x.Name ==
"Unipv").OrderBy(x => x.BlogId);

```

The ORM does all the work, mapping the database and reading/writing objects.


Many languages supports query language, like LINQ, which allow the developer to use a SQL-like approach querying objects and entities

<base16/> THINKING HEX > Process 20

We mentioned controlled, quality-oriented processes, what are they?


An idea would be:

- Enforcing a strict predictive PM methodology (PRINCE2, ...)
- Micro-managing every activity
- Assigning specific roles within the team
- Adopting some kind of QA (with a specific team)


 > Process 21

The project lifecycle would become something like:

- Gathering requirements
- Analysis
- Producing specs
- Project tasks, activities and milestones
- Development
- Test
- UAT
- Release

 > Process 22

The problem is that it simply does not work well in real world, because:



- It is not possible to predict in advance (several months) how the requirements will change (and they will, they always do);
- Business just moves faster than developers: what is needed today, will not be necessary tomorrow

<base16/> THINKING HEX > Process: Agile 23

AGILE DEVELOPMENT

Agility is...

VALUES

adaptability
transparency
simplicity
unity

STRATEGY: charter, funding

RELEASE: goals, vision

ITERATION: backlog, release plan, review

DAILY: standup, refactor, integration

CONTINUOUS: TDD, build, collaboration

Working Software

visibility
velocity
burnup
tests

ACCELERATE DELIVERY

Let's try something different:
let's go agile.

Idea: there is no need to predict and plan everything, but just what is needed to complete the next iteration.

Concept: Work is organized in iterations, meaning that a project is approached with a time-boxed method.

<base16/> THINKING HEX > An agile methodology: SCRUM 24

- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.

From: Mike Cohn, Mountain Goat Software LLC

<base16/>
THINKING HEX

> An agile methodology: SCRUM 25

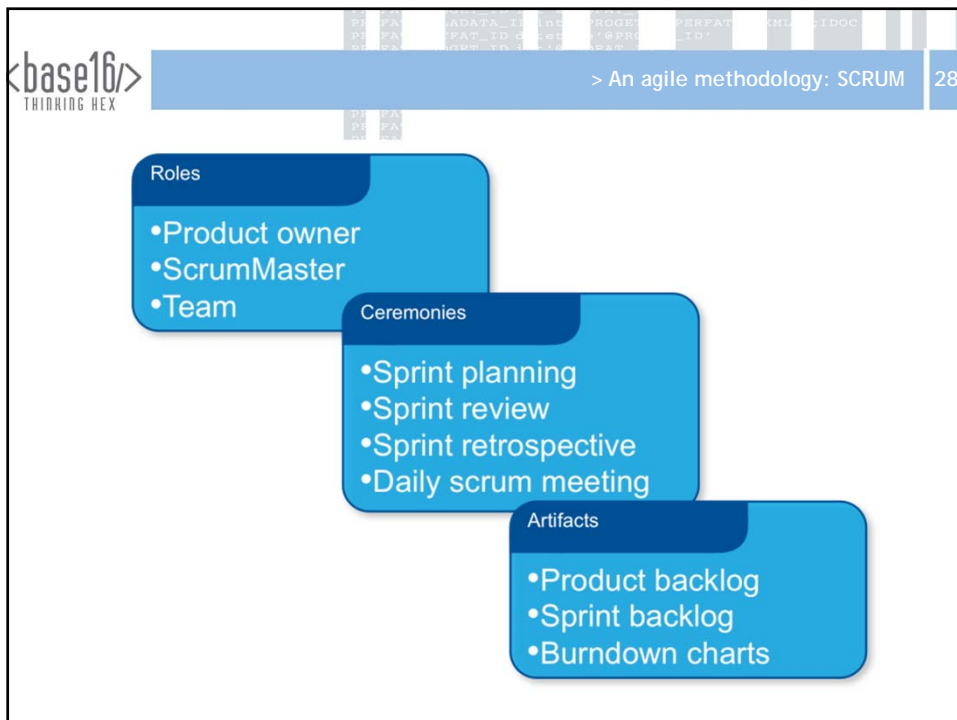
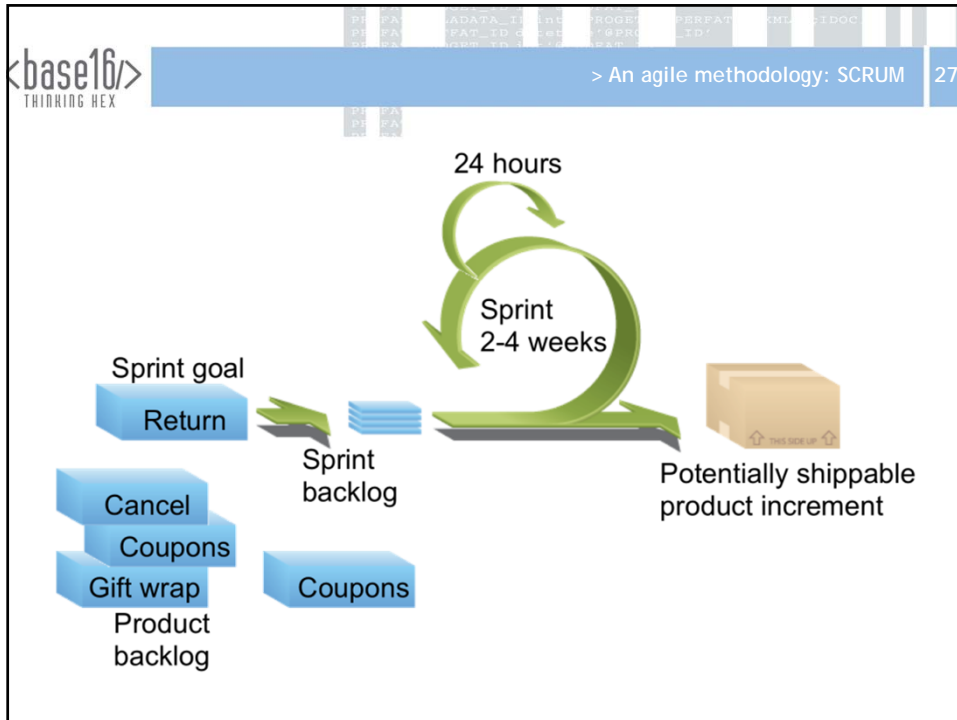
Main characteristics:

- Self-organizing teams
- Product progresses in a series of “sprints”
- Requirements are captured as items in a list of “product backlog”
- No specific engineering practices prescribed
- Uses generative rules to create an agile environment for delivering projects
- One of the “agile processes”

<base16/>
THINKING HEX

> An agile methodology: SCRUM 26

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan



<base16/>
THINKING HEX

> Bibliografia 29

Want to know more?

- Agile Software Development, Principles, Patterns, and Practices, Robert C. Martin
- [http://msdn.microsoft.com/en-us/library/ff847525\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/ff847525(v=vs.100).aspx)
- <http://www.scrum.org/>

<base16/>
THINKING HEX

PROFAT_PROJECT_ID datetime @PROFAT_ID'
OPAT_CREATE_ID datetime @CREATE_ID')
FROM OPENROW (CIDOC, 1/5000/P)
PROFAT_PROJECT_ID datetime @PROFAT_ID'
PROFAT_PROJECT_ID datetime @PROFAT_ID'
PROFAT_PROJECT_ID datetime @PROFAT_ID'
PROFAT_PROJECT_ID datetime @PROFAT_ID'
PROFAT_PROJECT_ID datetime @PROFAT_ID'
PROFAT_PROJECT_ID datetime @PROFAT_ID'

234567890 abcdef

Thank you for the attention