MASARYKOVA UNIVERZITA

PV213 Enterprise Information Systems in Practice

12 – Deployment, Migration, Maintenance

MASARYKOVA UNIVERZITA

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Deployment

Deployment is installation of the system on machines that makes system available for use.

- Deployment can be very simple (e.g. copying some files to one machine) or quite complex (special installation on several different machines)
  - Scripting and automation can help
- Deployment is risky operation especially when deployment is done on productive systems and there isn’t possibility to switch back to previous working version
  - Deployment must be handled carefully

Example of hard to do deployment - Introduction

- Real example from late 1990s and early 2000s
- System was Customer Relationship Management (CRM) which stores information about customers (contacts, meetings with customers etc.)
- The system was 2-tier architecture with central database and GUI on client's PC directly connected with database
- System had one nice feature (from point of view of users). It was possible to install the whole system (database and GUI) on user's (salesmen's) mobile device (notebook) and use it in disconnected manner (without connection to the central server). Data were stored just locally on the notebook in separate database instance.
- When user was again on the network, local database instance was synchronized with the central server
- Users were geographically "distributed" in Switzerland and Germany

Example of hard to do deployment - Root of problems

- Synchronization means
  - New/changed/deleted records created/changed/deleted on notebook were stored/deleted in central database
  - New/changed/deleted records created/changed/deleted in central database were stored/deleted in local database copy on notebook
- Such synchronization caused two problems
  - There can be conflicts when two or more users change the same data
    - Simple approach was chosen – latest wins (not ideal but because different salesmen communicated with different customers it worked)
  - There is big problem with deployment of the new version

Example of hard to do deployment - Deployment hell

- To simplify logic related with synchronization the same database schema must be installed on the server and on the notebook
- This means that when there was change in the database schema all notebooks were collected to one place and update was executed at the same time
  - You can imagine how problematic such logistic action was
  - For this reason change of the database schema was done occasionally
- Of course there exists solution to this problem
  - Support different schemas during synchronization
  - Ability to do the local database migration automatically when connected to the central server first time

Different environments I

During the whole process from development to the final deployment of system in the production there usually exists several environments.

Development environment

- Environment against which developers do the development
- Most “messy” environment (usually with partially inconsistent data)

Integration environment

- Environment in which developers can integrate and test how their components work together
- Data are still partially inconsistent
- Sometimes development and integration environment are “merged”
  - E.g. there exists just one database which is shared by all developers
- Can be used also for running automatic tests (or some special integration environment can be used for internal automatic tests)

Different environments II

Internal testing environment

- Environment where are done tests by internal quality team (testers)
- Should not be used by developers
- Data should be consistent
- Should simulate productive environment (amount of data etc.)
- There can be several different test environments (e.g. for performance testing)

Customer’s testing environment

- Environment in which selected customer’s users can test the new version
- Should simulate productive environment as much as possible

Productive environment

- Environment which is used by all end users
- It should be prohibited to do any changes directly by developers
  - Incorrect change can have catastrophic consequences
  - There is possibility for data leakage

Deployments in different environments I

Development environment

- No real deployment. Developers use their tools to run the system or its part on their development machine(s)

Integration environment

- There should be some rules for deployment to avoid a problem that one developer influence others
  - At least all appropriate developers must communicate their requests
  - There can be internal rule e.g. that deployment is done automatically twice a day

Internal testing environment

- System must be deployed the same way as it will be deployed in further environments to find possible problems e.g. with migration
  - Database scripts for changing structure of the database schema and scripts for data migration must be prepared
- Deployment should be done by someone else than by developers

Deployments in different environments II

Customer's testing environment

- The same rules as for internal testing environment
  - But all problems found during internal testing should be already solved
- Deployment must be communicated with customer

Productive environment

- Deployment must be known enough time ahead and communicated with customer, he must agree
  - Exact downtime must be known and short
- If any downtime is unacceptable another approach must be taken
  - E.g. there are two environments and in the first step only first is changed. Users can still use another environment. The whole process should be transparent.
- This deployment is critical and all possible risks should be eliminated (backups, alternate ways how to proceed, etc.)

Access rights to different environments I

Development environment

- Without any restrictions

Integration environment

- Without restrictions but access must be coordinated (and documented)
- One person can be responsible for deployments – he must be substitutable and available (can slow down development speed)

Internal testing environment

- Testers are responsible for this environment. Developers just prepare installation and migration scripts for testers
- Developers should not touch this environment directly but they should be allowed at least to have read access to see and analyze what is wrong otherwise finding problems is inefficient

Access rights to different environments II

Customer's testing environment

- One (substitutable) person should be responsible for doing deployments
- Access rights should be restricted but process of analyzing possible problem must be efficient (efficient communication is required)

Productive environment

- One (substitutable) person should be responsible for doing deployments
- Access rights must be restricted (e.g. because of law)
- There must be setup efficient process how end developers can analyze root of possible problems in the productive environment (should be in exceptional cases)

Environments for released and new version(s)

Till now we described just environments for new developed version. But we need to have the set of environments also for **all already released versions** which we support.

- For this reason try to minimize amount of supported versions
  - Costs for maintaining several versions are too high
- Ideally is to have just one released version in the productive environment and one currently developed new version
  - Developers must be able to fix possible bugs in the released version
  - Testers must be able to test fixes made for released version
  - All components of the system must be in the same state as released version (e.g. database with given database schema)

Application code migration I

Application code migration is an installation of the new code on the machine where code runs

- Code migration depends on the architecture which is used
  - For thin clients and 1-tier architecture migrations is easier
  - For thick clients and n-tier architecture migration is harder
- You can take different approaches to code migration
  - Manual migration
  - Automatic migration
  - Mixed (manual and automatic) migration
- Which approach will be taken depends on the project
  - For small projects doing full automatic migration can be overkill
  - For big projects you need mostly automatic migration

Application code migration II

Example of code migration for thick client and 3-tier architecture.

- Code migration on the client
  - Most probably automatic approach will be used
  - Client part of the whole solution detects that new version is available for download
  - New version is downloaded and installed
- Code migration on the application and database server
  - If number of servers is low manual change of the code is enough
  - If number of servers is higher some helper scripts are required
- You have to solve an issue what to do when some clients are active at the time when server code is migrated (e.g. all clients must disconnect)
- You have to consider whether you support several client versions

Application code migration III

Example of code migration for thin client and 3-tier architecture (typical web application).

- Code migration on the client
  - Done automatically when web server is migrated
  - You have just take care about caching (some pages or scripts can be cached in the client's browser or in some public caches)
- Code migration on the web, application and database server
  - Nice when you can migrate each part independent (can be problematic)
  - For nontrivial number of servers scripts for migration needed
  - You have to deal with downtime (maintenance window) of the server
  - If zero downtime is needed you have to solve an issue that two versions are running in parallel

Application data migration

Data structures used by application must be consistent with code

- When code is migrated data must be migrated as well (if there are new features which requires changes in data structures)

How to deal with data migration

- Data stored locally on the client must be migrated together with code migration
  - One part of the migration is migration of data
  - Code and data must be migrated atomically (all or nothing)
- Data stored on servers must be migrated in a way which doesn’t influence functionality of clients
  - There can be maintenance window when servers are not reachable (users must be informed)
  - If zero downtime is needed migration must be done in several steps

Maintenance costs of several versions

If several versions of the application must be supported at the same time maintenance costs are growing rapidly.

- You have to maintain several different versions of the
  - Client software
  - Server components
    - Web server
    - Application server
    - Database server
- You have to maintain different protocol versions between client and server
- You have to support user’s for different versions (user manual, hotline, etc.)

Try to minimize amount of used versions!

Closing the project

Project is usually "closed" when all functionality is delivered to the customer and there isn't plan for new functionality.

- Project can be split into several milestones/versions
  - Sometimes there can be delay (e.g. several months) between versions
    - This is difficult situation due to planning of team members (another work has to be available for team members)
- On the end of the project there can be organic decrease of team members as work is finalized
- Sometimes amount of work is decreased with upcoming milestones so amount of team members must be decreased as well
- On the project end project manager should write final project report
- Project is then "switched" to the maintenance phase

Closing the project - Experience workshop

On the end of the project is good to organize “**experience workshop**”

- There are summarized good and bad experiences from the project (retrospection)
- All members (even these left the project earlier) can learn from the workshop what to do better in next projects
- It is recommended that workshop is moderated by someone outside of the team who has experience with moderation
- On all projects I participated “communication” was recognized as the weakest point
  - Technicians are usually introverts
  - **Developing communication skills is very important in professional career**

Specifics of maintenance projects I

Purpose of the maintenance phase is to **support used application.**

- Solving technical problems during execution
  - HW or SW failures
  - Problems caused by integration with other systems
  - Problems caused by found SW bugs
    - Fixes in the code
    - Data changes caused by bad functionality of the application
- Modifying application according to changes in the company
  - Changes in company processes, company structure, etc.
  - This can include changes of data as well
- Adding a limited amount of new functionality

Specifics of maintenance projects II

- Team is massively reduced
  - Usually original project manager do this only for small amount of time or project management is delegated to the new person responsible for maintenance
  - It is recommended that at least two persons can do the maintenance to easy substitute each other in case of sick or vacancies. They don’t have to work for 100% of time for maintenance (they can work on other projects as well)
- Maintenance phase of the project can be quite long
  - For “standard” projects it can be 3-10 years
  - For special projects (industry, critical systems, etc.) even longer
  - When maintenance phase is long you can expect some fluctuation of people (it also means that some know-how is lost)

Specifics of maintenance projects III

- It has to be carefully monitored whether number and type of new requests is **beyond the standard maintenance phase**
  - If requests change a lot “philosophy” of the application it is dangerous that new functionality will be inappropriately “glued” without bigger redesign (influences quality and further maintenance costs)
  - In this case there should be a warning “red flag” that changes has to be done in more systematic way
  - This is often overlooked by maintenance people
- Sometimes when there are a lot of new requests and it is not possible to do them because of technology deficiencies of the currently used version new generation of the application is build

Long term projects I

Long term projects are projects where “main development” takes several years (several consequent versions are developed).

- The main challenge is how to keep key project members in the team
  - You cannot avoid “standard” fluctuation of people but as project manager you have to do what you can to motivate people
  - Project manager must be motivated as well by the higher management
  - Doing social events can join team members together especially when team is bigger and members cannot sit together on the one place (e.g. twice or three times a year)
  - Especially it is important to avoid a situation that several team members leave the team in a short time (new members must have enough time to get know-how from old members)

Long term projects II

- It is important to keep documentation up to date so new members can be easily introduced into the project
- It is important that architecture of the project is adopted when new (incompatible) requirements are coming
  - When changes are just “glued” to the original architecture it can lead to the increase of development and maintenance costs
  - All team members should be continuously educated when there are changes in the architecture to avoid misunderstandings
  - Appropriate tools can help to detect abnormalities in the code
  - Agile approach can help
- It is good to do experience workshops after each major version (e.g. once or twice per year)

Customer support I

Customer support is important part of the whole project contract.

- Some companies generates more income from customer support (post sales activities) than from the sold product itself
- Customer support can be general (if product is sold as standard product to lot of customers) or customer specific (if product is highly modified for customer’s purposes)
- Customer support can be divided into several levels
  - Bronze (standard) support
  - Silver support
  - Gold support
  - Platinum support (the best)
- Difference between levels is usually what and how fast customer will get from the support (e.g. for platinum support 24 hours a day, 7 days a week, 365 days a year)

Customer support II

- Standard customer support usually means that customer can get patches and new versions within specified period of time (one year). After this time support has to be renewed (paid again)
- Customer support is part of general Service Level Agreement (SLA) which covers much more (e.g. performance of the application etc.)
- For reporting issues use just one interface (one general contact) to avoid chaos (and easily substitute persons). This interface can include
  - Mailbox (app-support@xyz)
  - Phone number
  - Special web application for reporting issues

Děkuji za pozornost.

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ