

Practical Programmer

The methodology of the professional and scientific work,
Belgrade, Mathematical Faculty, May 2014

Dejan Vesić, <http://www.vesic.org>

or

4 / 5

The methodology of the professional and scientific work,
Belgrade, Mathematical Faculty, May 2014

Dejan Vesić, <http://www.vesic.org>

Disclaimer

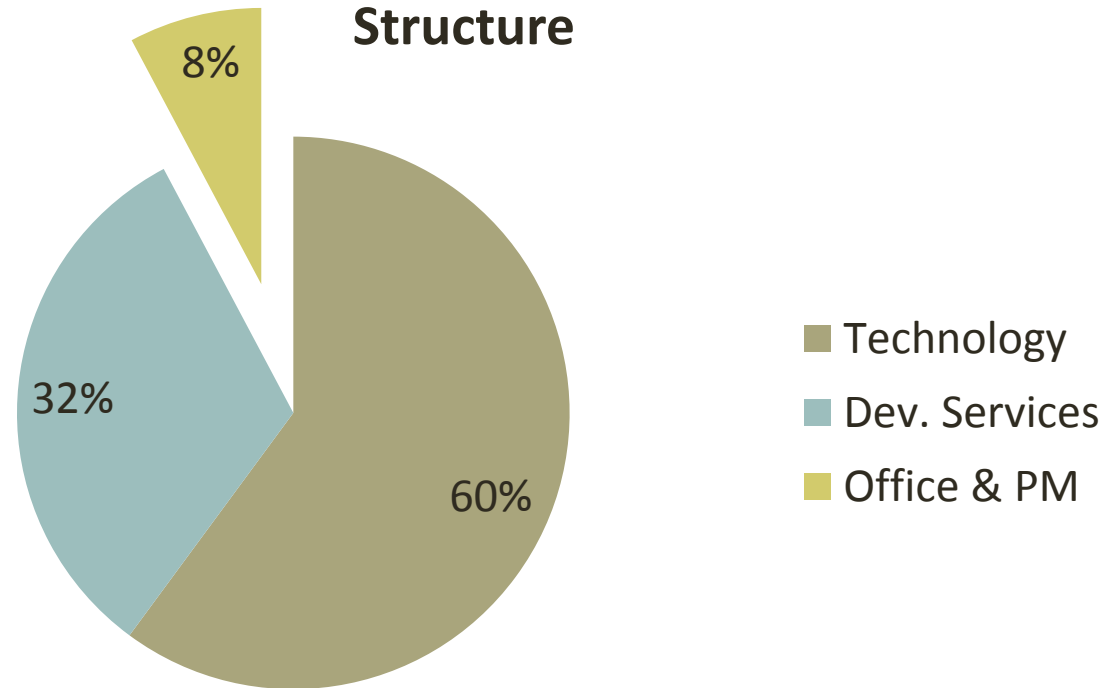
- Real life
- Happened many times (experience)
- This is about **Software Development** and **NOT** about **Computer Science**
- Honestly, I do not lie 😊
- I am old, probably outdated, but this is mandatory lecture ;-)

About me

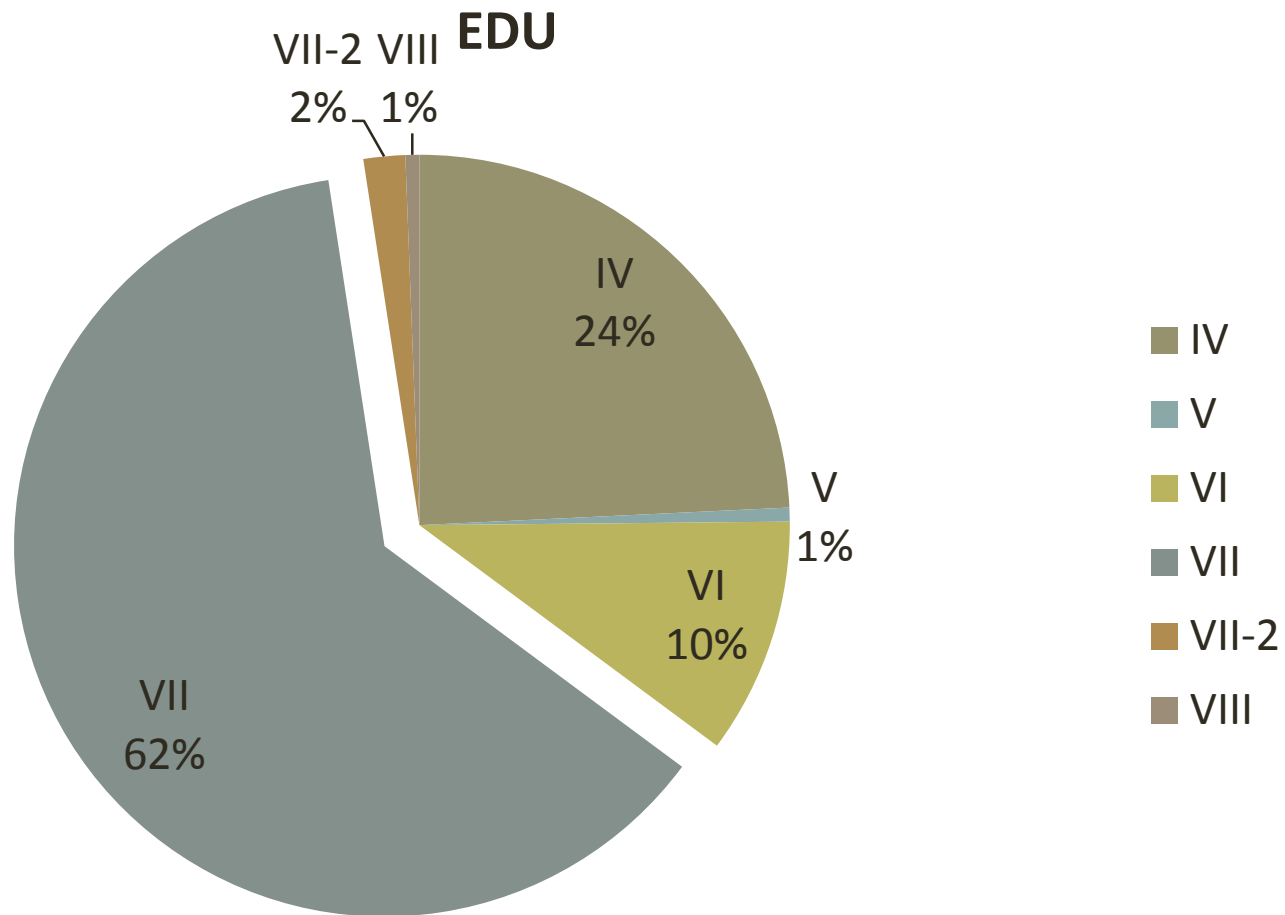
- Dejan Vesić
- Same age as Unix operating system (Bell Labs, 1969)
- More than 22 year in **commercial software development**
- Started as Clipper and FoxPro, then Oracle programmer, moved to Web (Asp & Asp.Net), ended on C# side (all rounder)
- Target systems: e-commerce sites of high risk and high traffic (bookmaking sites) backed by Oracle database
- Head of GTECH Belgrade company – more than 190 employees, of which 80+% are programmers (and 10 in UK)
- Member of IEEE Computer Society, ACM and Mensa
- More details on <http://www.vesic.org> (Serbian) and <http://www.vesic.org/english/>

GTECH Belgrade

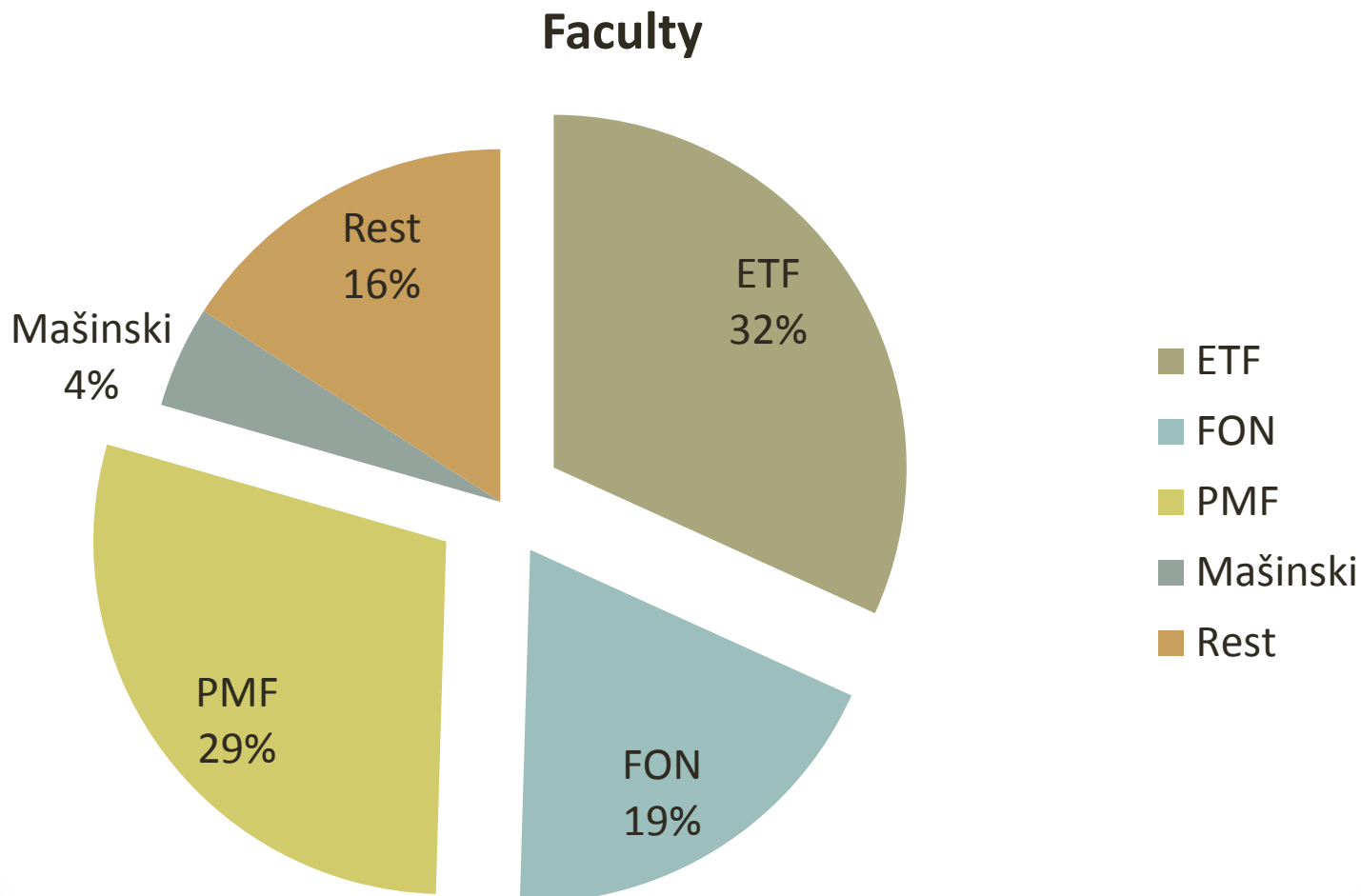
- From 1999 (BEG Finsoft)
- 190+ Employees: Techies - 150



GTECH Belgrade



GTECH Belgrade



Agenda

- Why this?
-
- You - Start of:
 - Identity
 - Communication
 - Work
 - First rule of Programming
 - You – Best Programmer Ever
 - Usual Problems
 - Dangerous Programming Errors
-
- Commercial Programming
 - Team Work
 - Documentation / Comments
-
- CV / Resume
 - CV: General Rules
 - CV: Bad Examples
 - CV: References
-
- Selection of Candidates
 - Interview - Preparation
 - Interview
 - Interview – About Money
 - Final Decision
 - Where Not to Look for Work
 - References

START

Start of You: Identity

- Mail address
 - Professional format (name.surname; no nicknames, or misleading terms)
 - Respect this medium as phone
 - Think twice – change is complicated
 - Private / Business one
 - Protect from spam (<http://www.sneakemail.com> or similar service)
- Presentation / Web page (online CV)
- Twitter / Facebook / LinkedIn
- Blog
- Discussion groups / forums
- **Build your personal brand – create something! ***

* <http://www.squidoo.com/distinguishyourself>

Start of You: Communication

- **Learn to communicate with:**
 - Customers
 - Clients
 - Users
 - Co-workers
 - Bosses
- Learn how to speak in public
- Learn how to persuade someone without shouting (guilty as charged 😊)
- Learn how to explain w/o jargon 😊
- **Learn to communicate**

Start of Work

- You only or in small team
- Even pro bono one (no money, but practice and references)
- Open Source projects (community, communication, team work, team tools, remote work; <http://sourceforge.net>)
- FreeLancer, Elance, Guru
 - Real Work – from requirements 'till support
 - Real Money – can be decent addition to income
 - Real Possibilities – can be start of very useful relations

Start of Work: Guidelines

- Keep learning (to avoid Coders syndrome)
- Learn a language (new one each 6 – 12 months)
- Read code of others – REGULARLY
- Use Design Patterns (but understand them!)
- DRY (Don't Repeat Yourself)
- Automate (Scripting languages / build systems)
- KISS (Keep It Simple Stupid 😊)

Bugs - First rule of Programming: You made bug!

- OS
 - Compiler
 - Third party Library
 - Programmer
-
- A diagram consisting of two curly braces on the right side of the list. The top brace groups the first three items (OS, Compiler, and Third party Library) and is labeled '7%'. The bottom brace groups the fourth item (Programmer) and is labeled '93%'.
- | Source | Percentage |
|---------------------|------------|
| OS | 7% |
| Compiler | |
| Third party Library | |
| Programmer | 93% |

You - Best Programmer Ever

- No one can't control (really) on what you are working
- Your boss can't make you to be good programmer
- **Only person** who can make you **great programmer is you**
- **Be Humble:**
"The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague."*
- **Be Vain:** make something that looks as good as it works

* <http://c2.com/cgi/wiki?TheHumbleProgrammer>
Edsger Dijkstra, 1972 Turing Award lecture

Daily Work

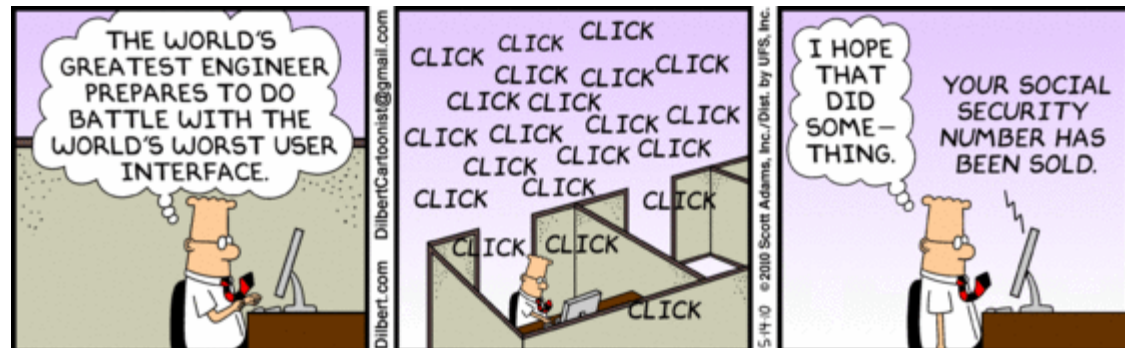
- Problems
 - Not Critical but important
- Errors
 - Critical
 - Can cause loss of money (for client) or private data
- Mistakes
 - Wrong ways to solve problems
 - Some out of your hands (team / process), some not

Usual problems (algorithms /techniques)

- Exceptions
- Object lifecycle (Disposable)
- Locking
- Multithreading
- Messaging
- Internationalization / localization (I18N)
- DB: Transactions, Locking, Triggers, Data reconciliation

Usual Problems (desktop)

- User Interface
 - Usability
 - All with keyboard as well
 - Distribution and order of controls on form
 - Adapt to environment (resolution / DPI / font size)
 - No confirmation for positive actions
 - Very careful selection of defaults
 - Standardization
 - As much as similar to existing applications on given platform (Office)
- Ease of Use
- Zero Tester (your mother? 😊)



<http://www.dilbert.com/>

Dilbert ©2010, United Feature Syndicate, Inc. All Rights Reserved

Usual Problems (Serbian ones)

- No responsibility (all excuses)
- Very bad code:
 - IPP (Serbian: APP)
 - Only positive branch
 - No standard way of error handling
 - Non-existent documentation
- Overdue delivery
- Over self-confidence w/o results behind that

2011 CWE/SANS Top 25 Most Dangerous Programming Errors (1/3)

- Insecure Interaction Between Components
 - Improper Neutralization of Special Elements used in an SQL Command ('**SQL Injection**')
 - Improper Neutralization of Special Elements used in an OS Command ('**OS Command Injection**')
 - Improper Neutralization of Input During Web Page Generation ('**Cross-site Scripting**')
 - Unrestricted **Upload** of File with **Dangerous Type**
 - Cross-Site Request Forgery (**CSRF**)
 - URL Redirection to Untrusted Site ('**Open Redirect**')

Reference: <http://cwe.mitre.org/top25/#Brief>

2011 CWE/SANS Top 25 Most Dangerous Programming Errors (2/3)

- Risky Resource Management
 - Buffer Copy without Checking Size of Input ('**Classic Buffer Overflow**')
 - Improper Limitation of a Pathname to a Restricted Directory ('**Path Traversal**')
 - Download of Code Without Integrity Check
 - Inclusion of Functionality from Untrusted Control Sphere
 - Use of Potentially Dangerous Function
 - Incorrect Calculation of Buffer Size
 - Uncontrolled Format String
 - Integer Overflow or Wraparound

Reference: <http://cwe.mitre.org/top25/#Brief>

2011 CWE/SANS Top 25 Most Dangerous Programming Errors (3/3)

- Porous Defenses

- Missing Authentication for Critical Function
- Missing Authorization
- Use of Hard-coded Credentials
- Missing Encryption of Sensitive Data
- Reliance on Untrusted Inputs in a Security Decision
- Execution with Unnecessary Privileges
- Incorrect Authorization
- Incorrect Permission Assignment for Critical Resource
- Use of a Broken or Risky Cryptographic Algorithm
- Improper Restriction of Excessive Authentication Attempts
- Use of a One-Way Hash without a Salt

Reference: <http://cwe.mitre.org/top25/#Brief>

35 Classic Mistakes*

People-Related Mistakes	Process-Related Mistakes	Product-Related Mistakes	Technology-Related Mistakes
1. Undermined motivation	14. Overly optimistic schedules	28. Requirements gold-plating	32. Silver-bullet syndrome
2. Weak personnel	15. Insufficient risk management	29. Feature creep	33. Overestimated savings from new tools or methods
3. Uncontrolled problem employees	16. Contractor failure	30. Developer gold-plating	34. Switching tools in the middle of a project
4. Heroics	17. Insufficient planning	31. Push me, pull me negotiation	35. Lack of automated source-code control
5. Adding people to a late project	18. Abandonment of planning under pressure		
6. Noisy, crowded offices	19. Wasted time during the fuzzy front end		
7. Friction between developers and customers	20. Shortchanged upstream activities		
8. Unrealistic expectations	21. Inadequate design		
9. Lack of effective project sponsorship	22. Shortchanged quality assurance		
10. Lack of stakeholder buy-in	23. Insufficient management controls		
11. Lack of user input	24. Premature or too frequent convergence		
12. Politics placed over substance	25. Omitting necessary tasks from estimates		
13. Wishful thinking	26. Planning to catch up later		
	27. Code-like-hell programming		

* This material is Copyright © 1996 by [Steven C. McConnell](#). All Rights Reserved.

Advices *

- Never stop **learning**.
- Do Programming ... a LOT!
- **Communication** is critical
- Learn how to WRITE in English (words, not code) – COMMUNICATION:
 - Blog
 - Active in community (any community)
 - Speak to real people
- Under promise, over **deliver**.
- Make stuff (working ones, for real people)
- "I was wrong."
- If it is not **tested** it doesn't work. If tested, it does not guarantee that it works what it should do
- Learn Microeconomics* (soon or later, it goes on money side)
 - And it helps to understand business

* <http://www.removingalldoubt.com/PermaLink.aspx/a32977e2-cb7d-42ea-9d25-5e539423affd>
„Fatherly Advice to New Programmers“, Chuck Jazdzewski

** <http://www.joelonsoftware.com/articles/StrategyLetterV.html>

Recommended Literature

- [Code Complete 2nd Edition](#)
Steve McConnell
- [The Pragmatic Programmer: From Journeyman to Master](#),
Andrew Hunt, David Thomas
- [The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition](#),
Frederick P. Brooks
- [24 Deadly Sins of Software Security](#),
Michael Howard, David LeBlanc, John Viega
- [Refactoring: Improving the Design of Existing Code](#)
Martin Fowler, John Brant, William Opdyke, Don Roberts
- [Rapid Development: Taming Wild Software Schedules](#)
Steve McConnell
- [Design Patterns: Elements of Reusable Object-Oriented Software](#)
Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides

COMMERCIAL PROGRAMMING

Desktop Application (example)

- Prerequisite check (and install)
 - OS version + service pack (check for minimal)
 - Mandatory OS components
 - Java runtime
 - .Net Framework
- Installation
 - As well under limited (non-admin) account
- Registration (machine signature)
- Desktop Application for something
- Error Logging
 - Remote Error Reporting
- Update
 - Backward compatibility (keep user data)
- Customer Support

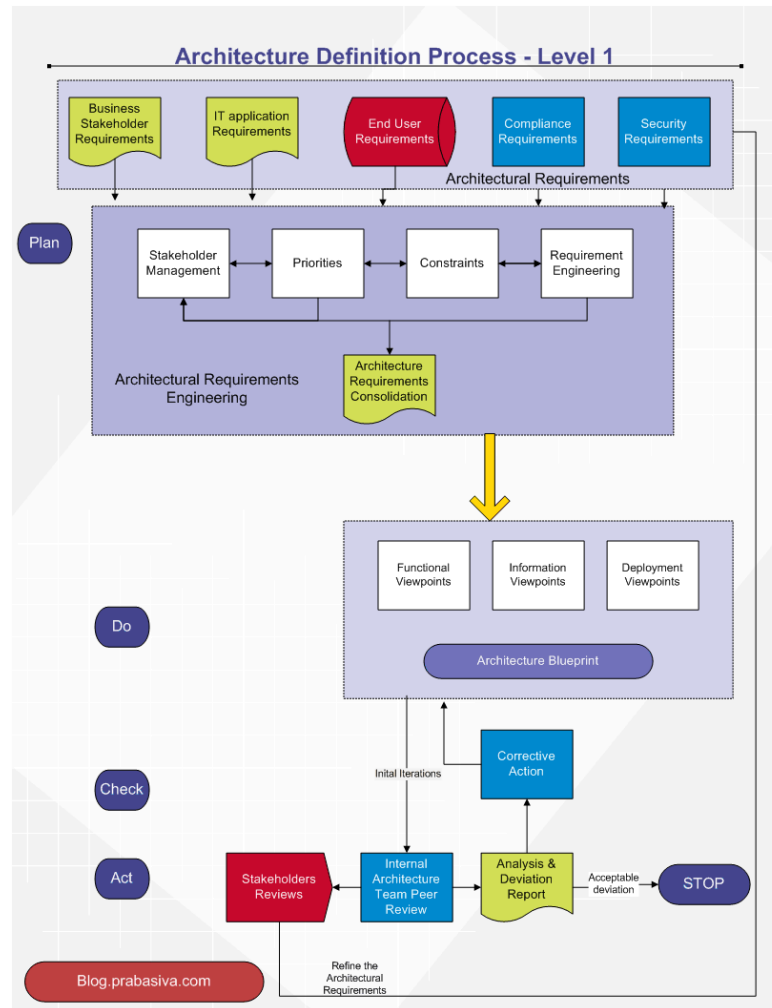
Commercial programming

- You **don't get paid to program, you get paid to ship**. Be good at your job^{*}
- Write software which will be used by someone (or that people will actually want to use)^{**}
- It's all about **what your code will do for the end user** and **not about how you did it**
- All Programming is Maintenance Programming

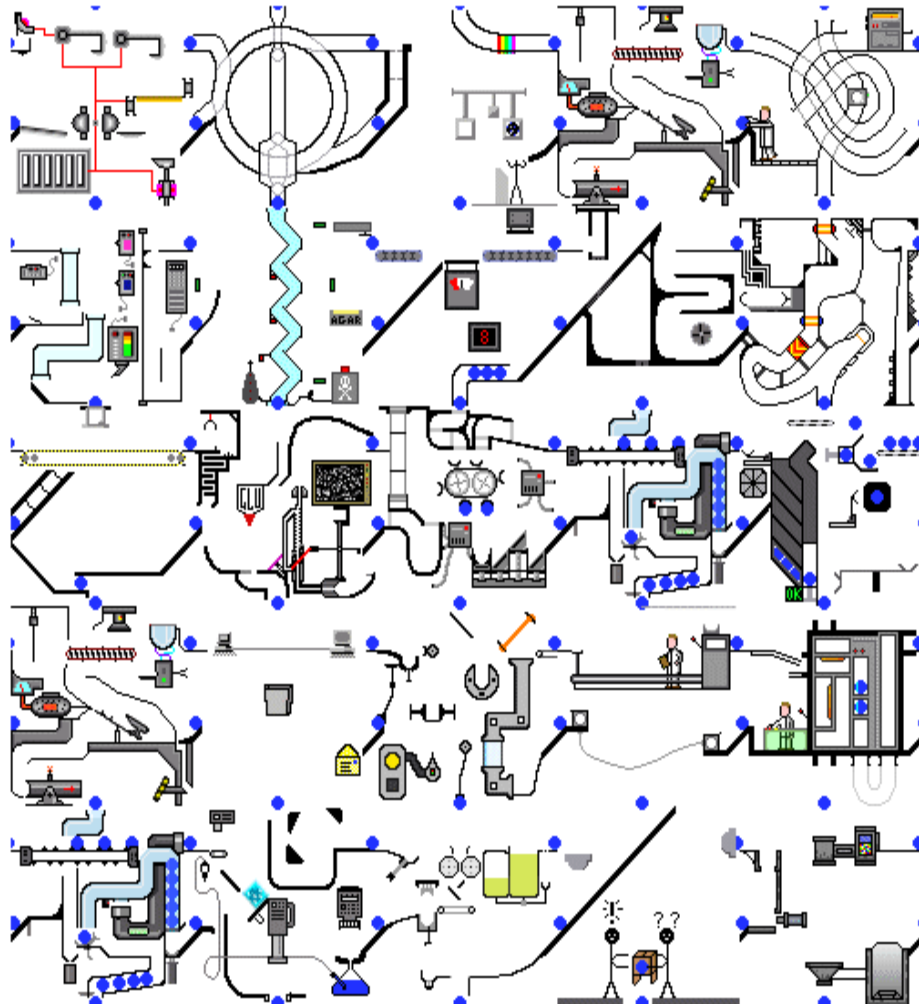
* <http://www.removingalldoubt.com/PermaLink.aspx/a32977e2-cb7d-42ea-9d25-5e539423affd> - Fatherly Advice to New Programmers, Chuck Jazdzewski

** http://www.skrenta.com/2007/01/market_engineering.html - How to Ship Code and Influence People, Rich Skrenta

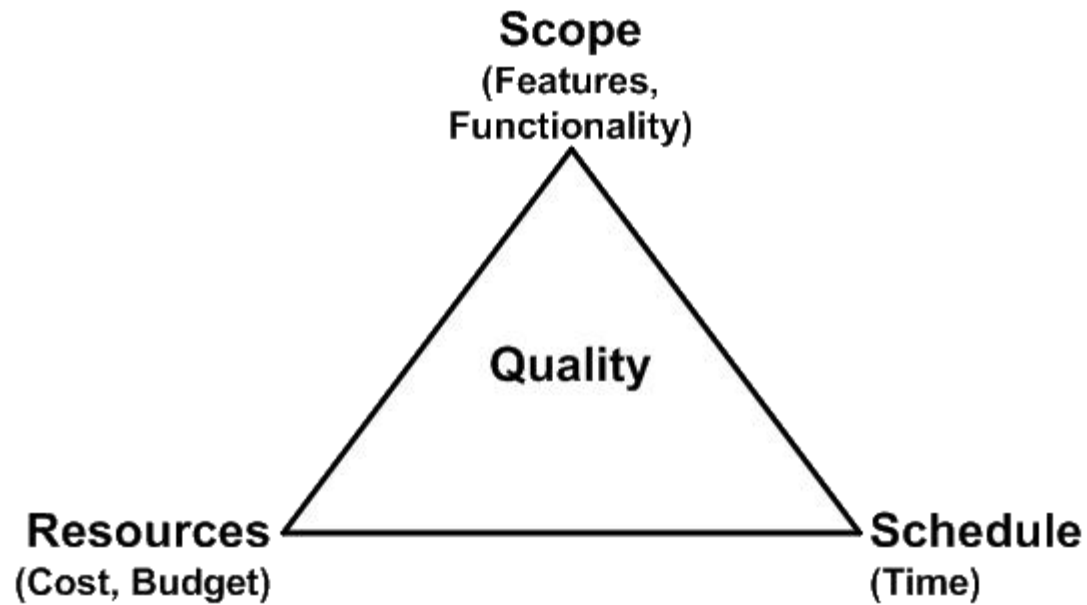
Software System



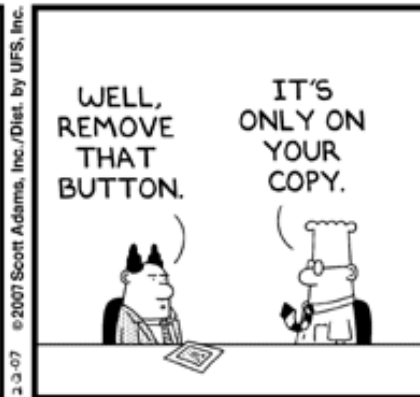
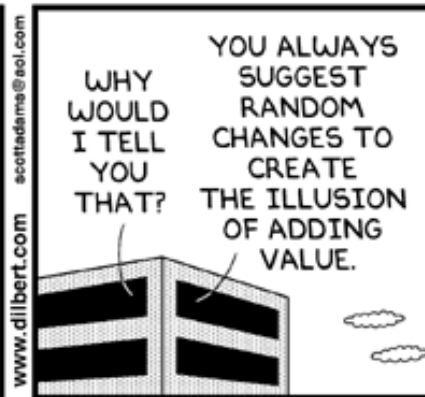
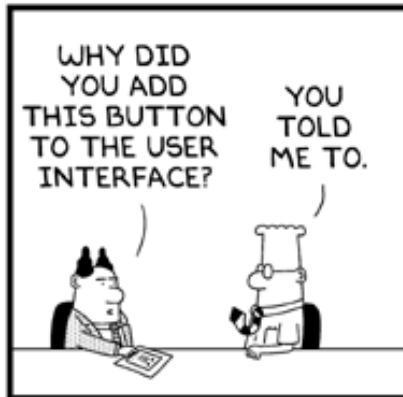
Real System



Iron Triangle



Real Problems 😊



© Scott Adams, Inc./Dist. by UFS, Inc.

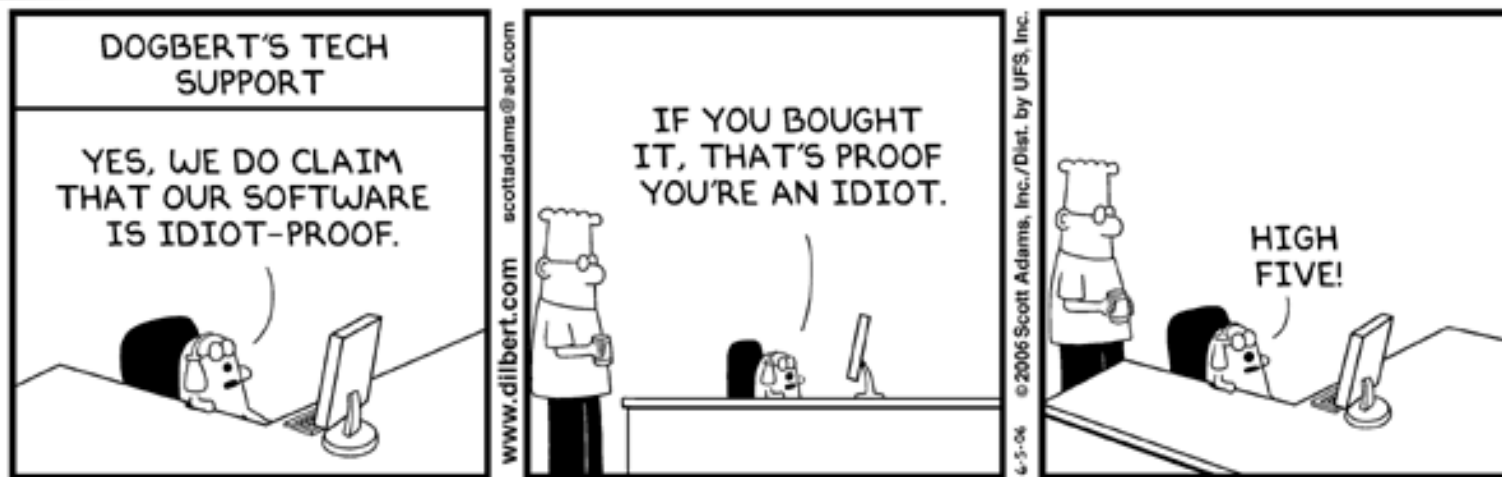


© Scott Adams, Inc./Dist. by UFS, Inc.

<http://www.dilbert.com/>

Dilbert ©2010, United Feature Syndicate, Inc. All Rights Reserved

More Real Problems 😊



© Scott Adams, Inc./Dist. by UFS, Inc.



© UFS, Inc.

<http://www.dilbert.com/>

Dilbert ©2010, United Feature Syndicate, Inc. All Rights Reserved

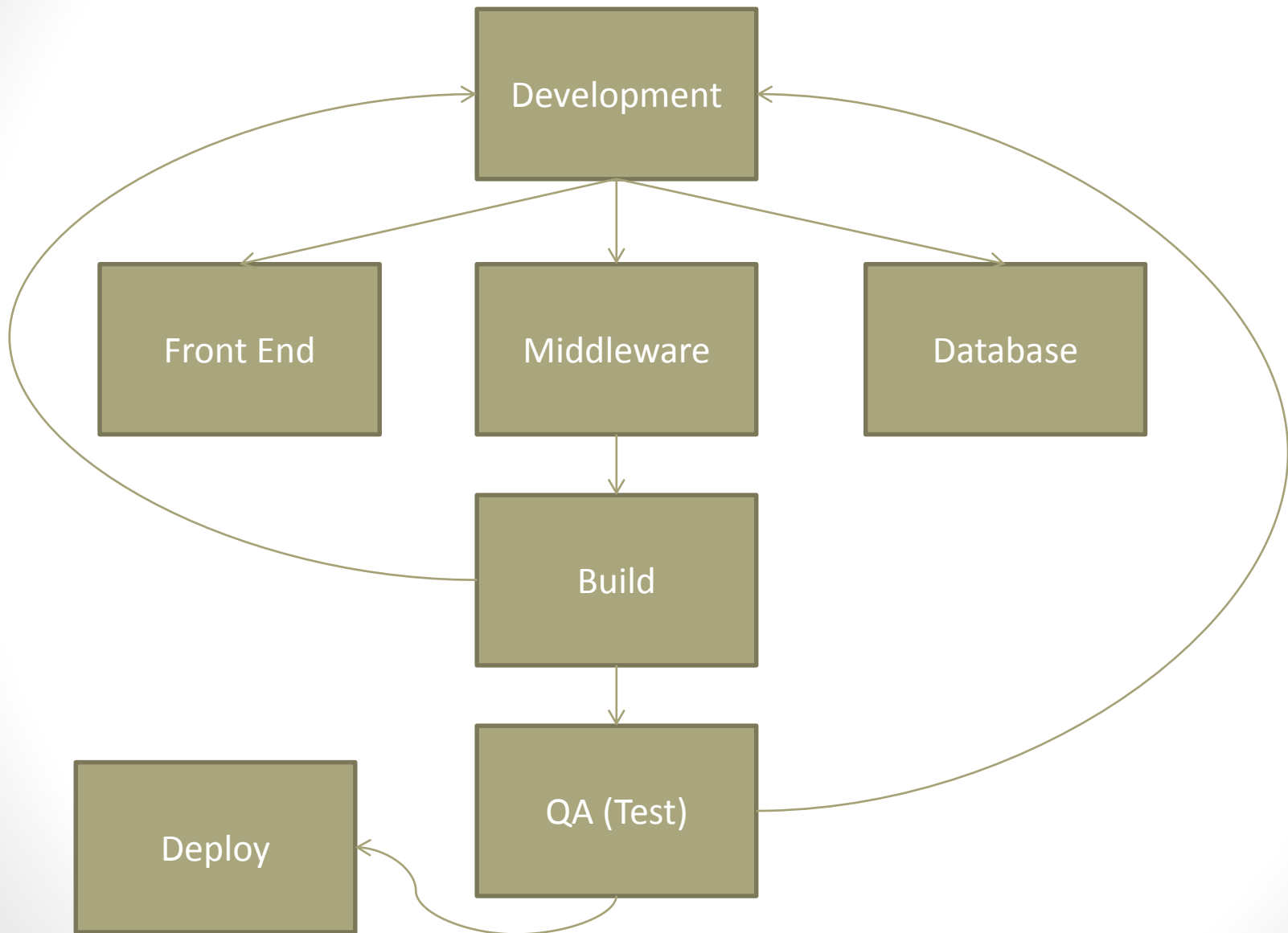
[Poll] Most important thing to deliver:

1. Operating system (Linux, Windows, OSX)?
2. Programming ideology (Commercial, Open Source)?
3. Language type (interpreted / compiled)?
4. Framework?
5. Specific language (Java, C, C++, C#, Ruby, Python ...)?
6. Right software development philosophy (Waterfall, XP, Agile, Scrum)? Right set of software development tools (Source Code Control System, Build system, Testing Framework ...)?

Most important thing to deliver:

TEAM!

Software Factory



How Good Your Team Is?

(technology)

- Do you use source code control system?
- Can you make a build in one step?
- Do you make daily builds?
- Do you have a bug database?
- Do you fix bugs before writing new code?
- Do you have an up-to-date schedule?
- Do you have a spec?

How Good Your Team Is For You?

- Do you respect your coworkers?
- Do you like your coworkers?

Teams – Real Life Problems

- Personality Problems
- Distributed Team (communication issues)
- Split Responsibility (unknown owner)
- Duplication of code / functionality (missing design)
- Knowledge Sharing (Wiki / Stack Overflow / Bug Tracking tools)
- Clear lines of Reporting
- Noise

Documentation

- High Level
- Structured
- Explains details which are higher than code
- Targeted toward human, not toward compiler
- Types:
 - Requirements
 - Design
 - Technical (algorithms, interfaces, APIs) - default
 - End User (tutorial, thematic, list or reference doc)
 - Marketing
- „Prior, clear, and extensive documentation is a key element in creating software that can survive and adapt“*
- Documentation is COMMUNICATION

* <http://queue.acm.org/detail.cfm?id=1053354> - Comments are More Important than Code

Documentation (example)

- What is it?
 - Why is written?
 - How it works?
 - Limitations
- Basic Context (where to use it)
- Installation
 - Upgrade
- Configuration
- Examples
- Troubleshooting
- Licensing / Copyright issues
- Good example of documentation:
<http://www.urlrewriting.net/160/en/documentation.html>

Comments

- Write extensive comments
- Write comments before code itself
- Comment even inline code
- Keep revision history in header of file
- Use auto-generated documentation

Writing comments:

- Is boring in a first place
- Takes time out of coding time
- Easy to forget to update when signature changes
- „When I wrote this, only God and I understood what I was doing.
Now, God only knows“ *

* **Karl Weierstrass, mathematician**

Examples of Comments ☺

- `// Magic. Do not touch.`
- `/* You are not meant to understand this */`
- `// drunk, fix later`
- `return 1; # returns 1`
- `// I'm sorry.`
- `// I am not sure if we need this, but too scared to delete.`
- `// I am not responsible of this code.`
`// They made me write it, against my will.`
- `/*`
`* You may think you know what the following code does.`
`* But you don't. Trust me.`
`* Fiddle with it, and you'll spend many a sleepless`
`* night cursing the moment you thought you'd be clever`
`* enough to "optimize" the code below.`
`* Now close this file and go play with something else.`
`*/`
- <http://code.google.com/p/xee/source/browse/trunk/XeePhotoshopLoader.m?spec=svn28&r=11#107>

Comments: HOWTO

- MAKE CODE SELF EXPLANATORY (so that you do not need to write comments) by using:
 - Same coding standard across team
 - Good variable names
 - Write / re-write / refactor code so that speaks for itself
- Use comments to **communicate ideas** to other **HUMAN BEINGS***
- Good comments -> you have to be good writer
- Good comment **answer on "Why"** (... is this algorithm / idea used) **and not on "What"** (... is going on in code) or "How" (.. is done)

* <http://www-cs-faculty.stanford.edu/~knuth/lp.html> - Literate Programming, Donald E. Knuth

Comments: Example

Initial:

```
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
    r = 0.5 * ( r + (n/r) );
}
System.out.println( "r = " + r );
```

First:

```
// square root of n with Newton-Raphson
approximation
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
    r = 0.5 * ( r + (n/r) );
}
System.out.println( "r = " + r );
```

Final:

```
private double SquareRootApproximation(n) {
    r = n / 2;
    while ( abs( r - (n/r) ) > t ) {
        r = 0.5 * ( r + (n/r) );
    }
    return r;
}
System.out.println( "r = " + SquareRootApproximation(r) );
```

CV

CV, Hiring, Interview – Disclaimer!

- Strictly personal view
- Based on previous personal experience
- Take this “as is” – without any warranty that it will work with other employer
- I **don't** want to say “this is right approach” but this is **my** current approach!

How to present yourself

- How to write CV / Resume
- Cover letter (application letter)
- Examples 😊
- Recommendation letters
- References

CV != Resume

- There are several differences between a curriculum vitae and a resume.
- A Curriculum Vitae is usually longer (two or more pages) than resume
- When asking for a job in Europe, the Middle East, Africa, or Asia, expect to submit a CV rather than a resume.
- Some of personal information on a curriculum vitae that would never be included on an American resume, such as date of birth, nationality and place of birth.
- United States law on what information job applicants can be asked to provide does not apply outside the country.

CV (1)

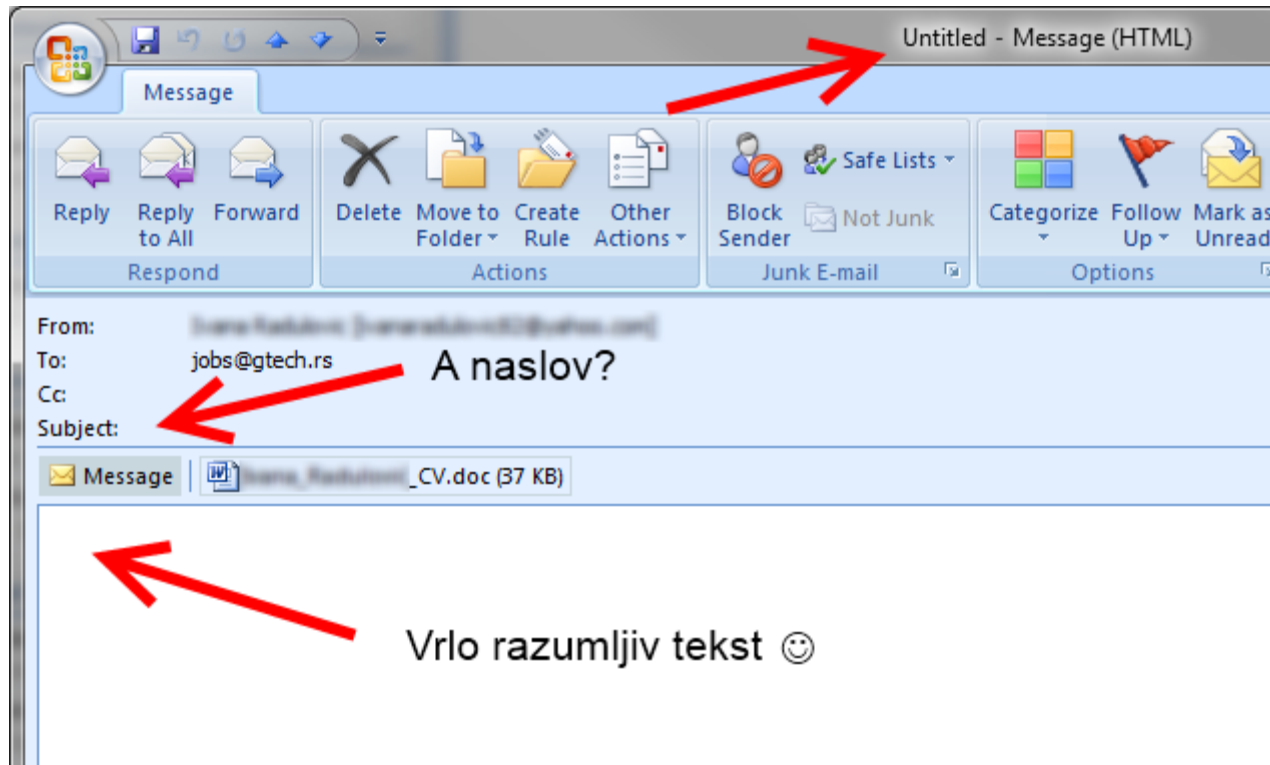
- Curriculum vitae should include:
 - your name
 - contact information
 - education
 - skills
 - experience.
- In addition to the basics, a CV includes:
 - research and teaching experience
 - publications
 - grants and fellowships
 - professional associations and licenses
 - awards
 - and other information relevant to the position you are applying for.
- Start by making a list of all your background information, then organize it into categories. Make sure you include dates on all the publications you include.

CV (2)

(practicalities)

- CV
 - Universal format (PDF)
 - If not PDF, make it as an archive (7-zip, Zip)
 - CV according to job you are applying (emphasize elements important for that position)
 - Content:
 - No more than two page
 - No grammar or spelling errors
 - Live references (work – sites and/or proper mail of your contacts)
 - **UP TO DATE!**
 - **UP TO DATE!**

CV – Bad (1)



CV – Bad (2)

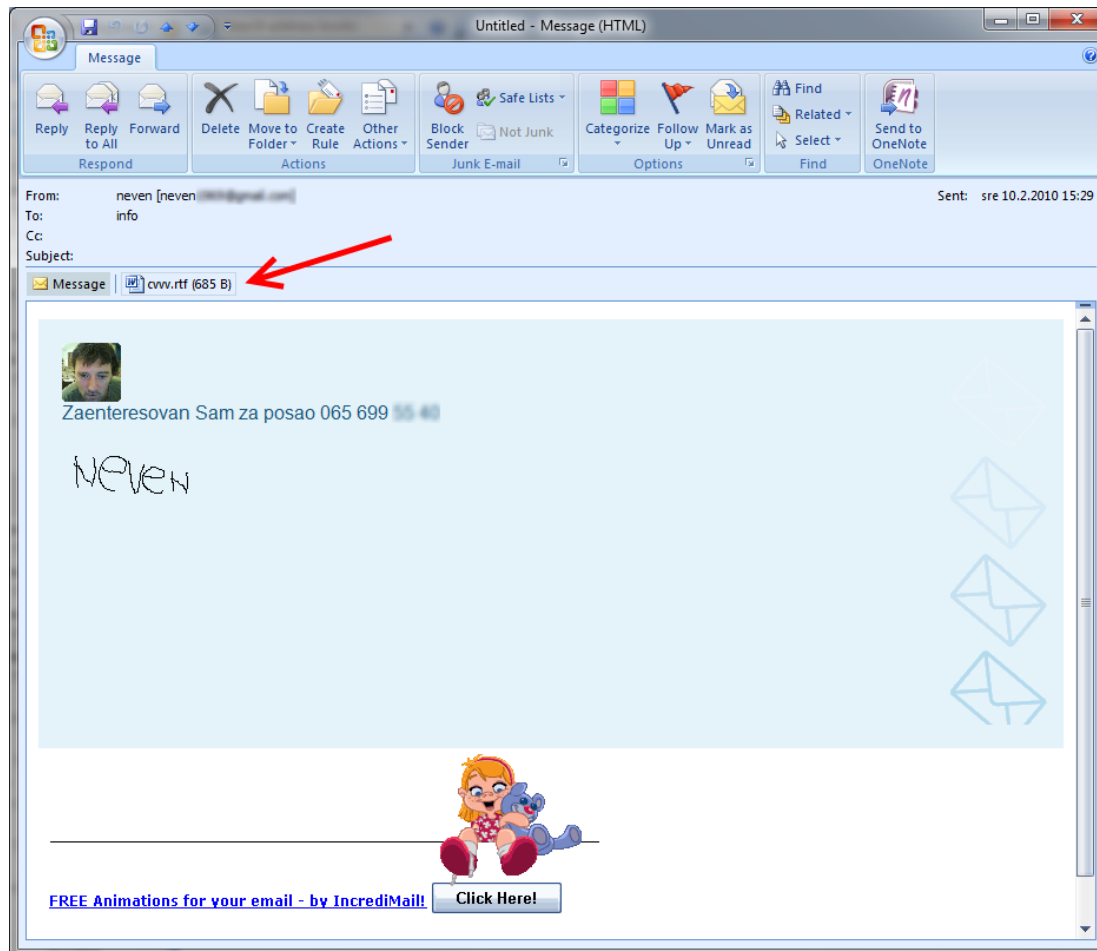
From: Stasa [\[stasaaa@gmail.com\]](mailto:stasaaa@gmail.com)
To: gtech-jobs@gtech.rs
Cc:
Subject: prijava za posao programera

Prijavljujem se za posao programera.....
Broj mog telefona je 064 - 4385 - 222

puno pozdrava.....

Stasa [\[stasaaa@gmail.com\]](mailto:stasaaa@gmail.com)

CV – Bad (3: Mail)



CV – Bad (3: Attachment)

12/14/2009

cv ~~XXXXXXXXXX~~ Neven

~~XXXXXXXXXX~~ NEVEN IZ KIKINDE

ZANIMANJE - IT TEHNISON

SERVISER RACUNARA
I MREZA I PROGRAMER

A+ SERTIFIKAT

o65699~~XXXXXXXXXX~~

CV -> JOB

Checking references and recommendations

- References and recommendations are very important
 - Professors / Teachers
 - Former Bosses
 - Former Colleagues
 - Users of your previous products / services
- Those gets checked and people get contacted
- Programming community (in Serbia) is small – make note of that

How to apply for work?

- Preparation
 - Get informed about potential employer: web site, products, structure and public image
 - Be sure to know for what you are applying to – if there is not enough info in advertisement, dare to ask for more
- Application (Cover) Letter
 - From your mail address
 - On intended address
 - Make sure that note position you are targeting for
- Recommendations
- References

Selection of candidates based on CV

- How it really works (in practice)?
- Sometimes just quick overview of CV (few seconds), sometimes very detailed
- Criteria
 - Internal (from company) recommendations and information
 - Skills
 - Ability to Learn (stuff already done)
 - Personal (team working, communication, languages...)
 - Experience
 - Education
 - External info

Interview - Preparation

- Get informed (good!) about potential employer
 - Official sources (web sites, materials, search, financial records)
 - Unofficial sources (current or ex-employees etc.)
- Get your (minimal) terms under you would accept position with that company
- Clear idea for which position you are applying to and under which conditions
- Prepare list of not-so-comfortable list of questions (for potential employer) as well list of your answers on similar questions

Interview

- Usually: Two parts + two circles
 - Personality and team member roles
 - Expert (for area of expertise required for specific role)
- Sometimes very informal
- Interview is bidirectional – be prepared to ask, not just to be asked
- Expect pressure and be prepared to it
- Be ready to say „No“

Interview – About Money

- Money is very important factor (but not only one!)
- Know price of your work
- Do not be ashamed to ask that price
- Prepare list of minimal conditions which you expect employer to fulfill
- Be sure that both sides are fully aware about agreed conditions! (repeat and verify before leaving final discussion and making decision)

Final decision

- Huh... this is hard to explain
- We consider all previously noted elements:
 - CV
 - All conversations
 - View of other colleagues involved in your interview
- If we can't decide... sometimes we follow our intuition

Where to look for work



(SugarCRM, Openbravo ERP,
PSTextil ERP)



Where not to look for work

- The leading association in the systems for payment online is looking for:
The senior programmer with at least 5 years of experience. The candidate has to have a **complete and total knowledge** of the **programs languages** which have been **indicated**, and has to prove that he can handle these projects alone. The candidate has to deal with European and American reality for that reasons he should speak and write fluently english. Experience in the sector of mobile telephones is the most important advantage.

The program languages (5 years experience)

- HTML, XHTML, DHTML / • XML, XSL, XSLT, DOM.
- CSS1 & 2, Javascript. / • Microsoft ASP (ADO2 & ADO3), ASP.NET
- PHP 3, 4 & 5. / • Sun Java Server Pages (JSP & Servlet).
- MySQL, Microsoft Access & Microsoft SQL Server 2000

The candidate should also have the good knowledge of the things such as -web server, ftp and mail server, nntp

- Microsoft IIS4 & 5. / • Linux
- Apache v1.XX & v2.XX. / • Apache Tomcat 4.XX

(real job offer, December 2008)

References

- Steve McConnell, <http://www.stevemcconnell.com/>
- Joel Spolsky, <http://www.joelonsoftware.com/>
- Scott Guthrie, <http://weblogs.asp.net/scottgu/>
- Jeff Atwood, <http://www.codinghorror.com>

- Dejan Vesić,
 - <http://www.vesic.org/matfplus/> 😊
 - Twitter: [@Vesic](https://twitter.com/Vesic)
 - LinkedIn: <http://rs.linkedin.com/in/dejanvesic>

Q & A

?