**Ieteikumi paskaidrojošam tekstam (J.Borzovs, 2019.; papildināts 2020.)**

**Šis materiāls neskar programmatūras prasību specifikāciju (software requirements specification), programmatūras projektējuma aprakstu (software design description), pirmkoda (source code) komentārus un testēšanas dokumentāciju, jo šajos aspektos parasti domstarpību nav. Turpmākajā tekstā skaidrojam programmatūras projekta organizāciju, programmatūras kvalitātes nodrošināšanu (software quality assurance), programmatūras konfigurāciju pārvaldību un darbietilpības novērtēšanu (precīzāk – gatavas programmatūras izstrādes normatīvās darbietilpības pamatošanu).**

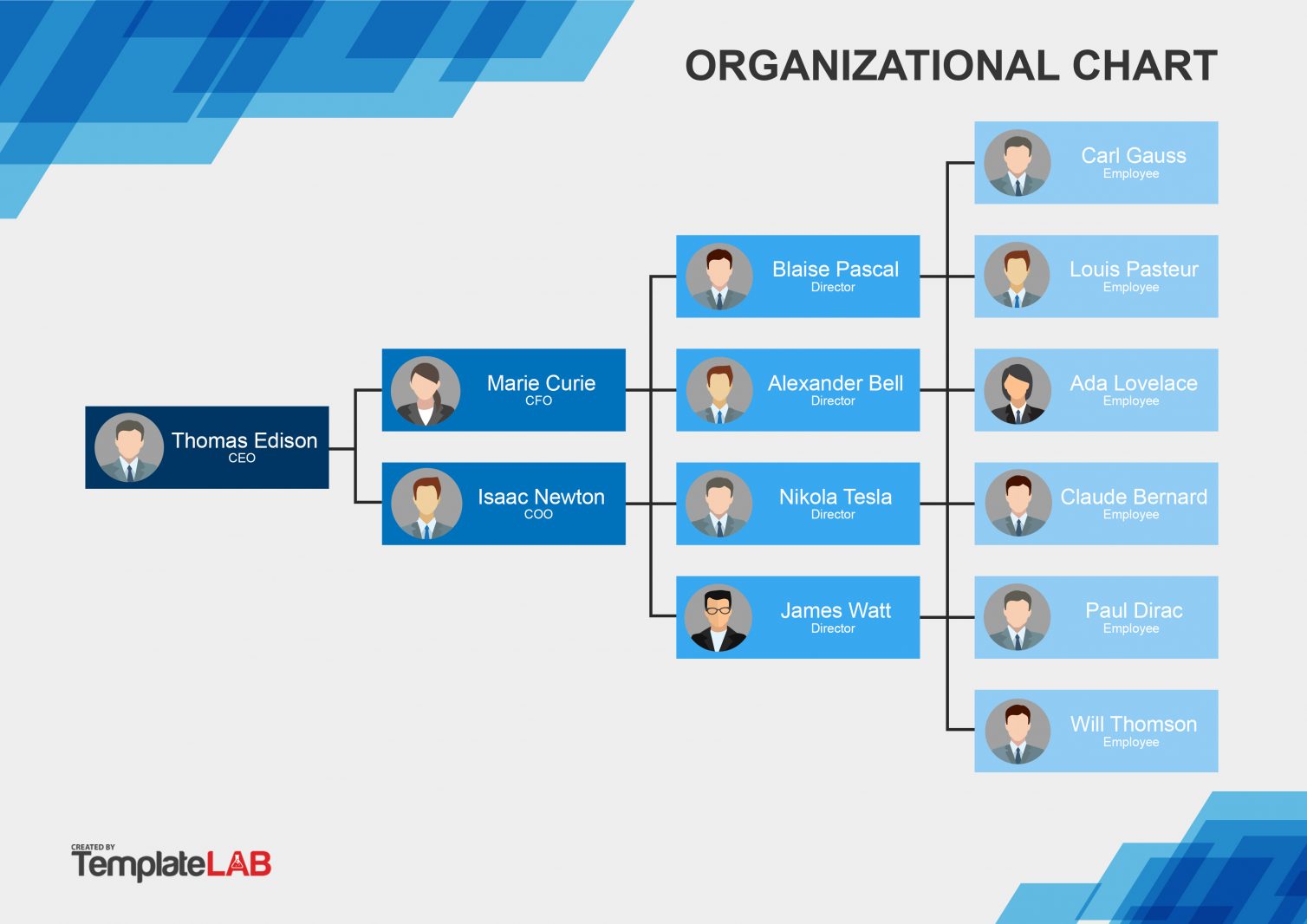
1. **Programmatūras projekta organizācija**

Ar šo jāsaprot attiecības starp projektā iesaistītiem darbiniekiem gan projekta grupas (project team) ietvaros, gan ārpus tās.

Atsvaidziniet savas zināšanas, pārlasot [ 1 ] CHAPTER 24: Project Management Concepts: People: The Software Team, 651-454.

Bagātīgu projekta organizācijas grafisku attēlojumu klāstu var redzēt <http://templatelab.com/organizational-chart-templates/> .

Vienu no daudziem iespējamiem piemēriem (hierarhiska organizācija, aizvērtā paradigma – closed paradigm) skatiet tepat.



1. **Kvalitātes nodrošināšana**

Atsvaidziniet savas zināšanas, pārlasot [ 1 ] CHAPTER 16: Software Quality Assurance, 432-448 . Atceramies, ka runa pamatā ir par projekta līmeni.

Angliskajam vārdam ‘assurance’ ir vairākas nozīmes. Dotajā gadījumā – gan ‘panākt, lai programmprodukts būtu kvalitatīvs’, gan arī ‘apliecināt pasūtītājam, ka izpildītājs dara visu nepieciešamo kvalitatīva produkta izstrādei’.

Programmatūras kvalitātes nodrošināšana ietver vairākas sastāvdaļas (R.S.Pressman):

1)      kvalitātes pārvaldības pieeja (quality management approach; organizācijas līmenis - JB);

2)      rezultatīva programminženierijas tehnoloģija (metodes un rīki);

3)      formālās tehniskās apskates (formal technical reviews, FTR) visā programmizstrādes procesā;

4)      daudzkārtējas (multitiered) testēšanas stratēģija;

5)      programmatūras dokumentācijas un tās izmaiņu vadība;

6)      procedūra, kas panāk atbilstību (compliance) programmizstrādes standartiem (kad tas ir piemērojams);

7)      mērīšanas un ziņošanas (reporting) mehānismi.

Līdzīgs skats uz kvalitātes nodrošināšanu saskatāms vēsturiski pirmajā programminženierijas standartā ANSI/IEEE Std. 730-1984, kura sadaļu nosaukumus citējam zemāk.

I.      Purpose of Plan

II.     References

III.    Management

        1.      Organization

        2.      Tasks

        3.      Responsibilities

IV.    Documentation

        1.      Purpose

        2.      Required software engineering documents

        3.      Other documents

V.     Standards, Practices, and Conventions

VI.    Reviews and Audits

VII.   Test

VIII.  Problem Reporting and Corrective Actions

IX.    Tools, Techniques, and Methodologies

X.     Code Control

XI.     Media Control

XII.    Supplier Control

XIII.   Records Collection, Maintenance, and Retention

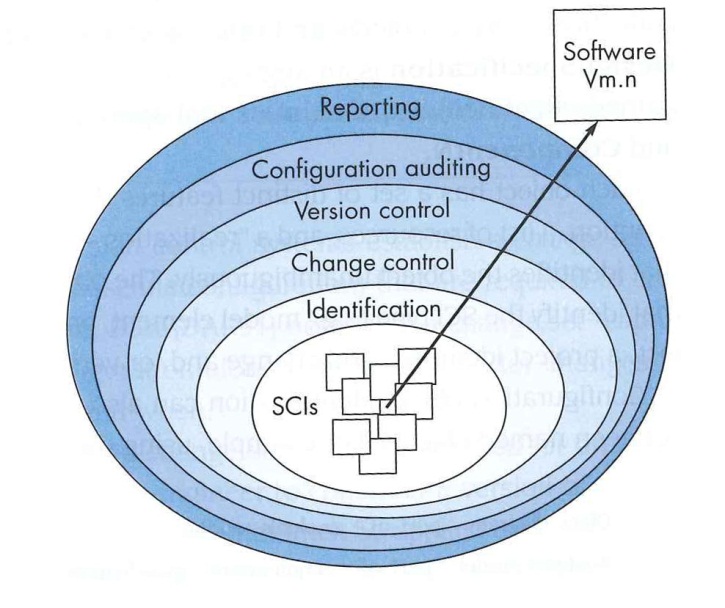
XIV.  Training

XV.   Risk Management

1. **Konfigurāciju pārvaldība**

Atsvaidziniet savas zināšanas, pārlasot [ 1 ] CHAPTER 22: Software Configuration Management, 584-612.

Atmiņas atsvaidzināšanai zīmējums no R.S. Pressman’a.



1. **Darbietilpības novērtējums**

**Pareizāk būtu teikt – izstrādātā programmkoda “normatīvās” darbietilpības pamatojums**.

Kā pamatot, ka programmkods ir vismaz 3 personmēnešu vērts ?

Pasaulē atzītākā komercsabiedrība, kas nodarbojas ar programmatūras darbietilpības prognozēšanu (effort estimation), ir QSM (Quantitative Software Management) - <http://www.qsm.com/> .

Jau izstrādāta programmkoda normatīvās darbietilpības noteikšana (nejaukt ar konkrētajā gadījumā faktiski patērēto) ir darbietilpības prognozēšanai inversa darbība. Programmkoda rindiņu skaits un/vai faktiskais funkcijpunktu daudzums ir zināms, bet jānoskaidro, cik profesionāļu cik ilgā laikā šo programmkodu būtu varējuši izstrādāt. Saprotams, ka to var pateikt, vien balstoties uz liela skaita projektu datiem. QSM regulāri krāj informāciju par pabeigtiem projektiem un rēķina minimālās, maksimālās, vidējās vērtības un mediānas. To pašu dažkārt dara progresīvas programmizstrādes komercsabiedrības, tāpēc ir vērts noskaidrot, vai konkrētā gadījumā ir pieejami komercsabiedrības dati, kurus vēl papildus varētu izmantot aprēķiniem.

Viena no labām iespējām ir izmantot QSM atklātā piekļuvē izliktās etalontabulas (benchmark tables) - <http://www.qsm.com/resources/qsm-benchmark-tables> , veicot saprātīgas interpolācijas vai ekstrapolācijas.

Piemēram, tabulas “Business Systems Implementation Unit (New and Modified IU) Benchmarks” aile Quartile 1 rāda informāciju par 25% mazākajiem (t.i., pirmās kvartiles jeb ceturtdaļas) no 550 projektiem. Šādi projekti vidēji ilguši 3,2 mēnešus, vidēji prasījuši 965 personstundas, tos izstrādājuši vidēji 1,57 cilvēki, šādu projektu lieluma mediāna ir 1889 programmkoda loģiskās rindiņas, bet projektu lielums variējais starp 131 u 3115 loģiskām rindiņām (vienā programmkoda rindiņā mēdz būt vairākas instrukcijas, bet var arī viena instrukcija aizņemt vairākas rindiņas; tukšās un komentāru rindiņas netiek uzskaitītas).

Ja Jūsu izstrādātā programmkoda lielums ir tuvs mediānai – 1889 -, recenzentam būs grūti pamatot, ka izstrādātā programmkoda normatīvā darbietilpība nesasniedz 3 personmēnešus. Protams, ka īpaši sarežģīta programmkoda gadījumā lielums būs pamatoti mazāks.

**Reference** (grāmata legāli pieejama internetā)

1. Roger S. Pressman. Software Engineering: A Practitioner’s Approach, 7th ed., 2010.