Presentation
Paper
Bio
Return to Main Menu

PRESENTATION

T12

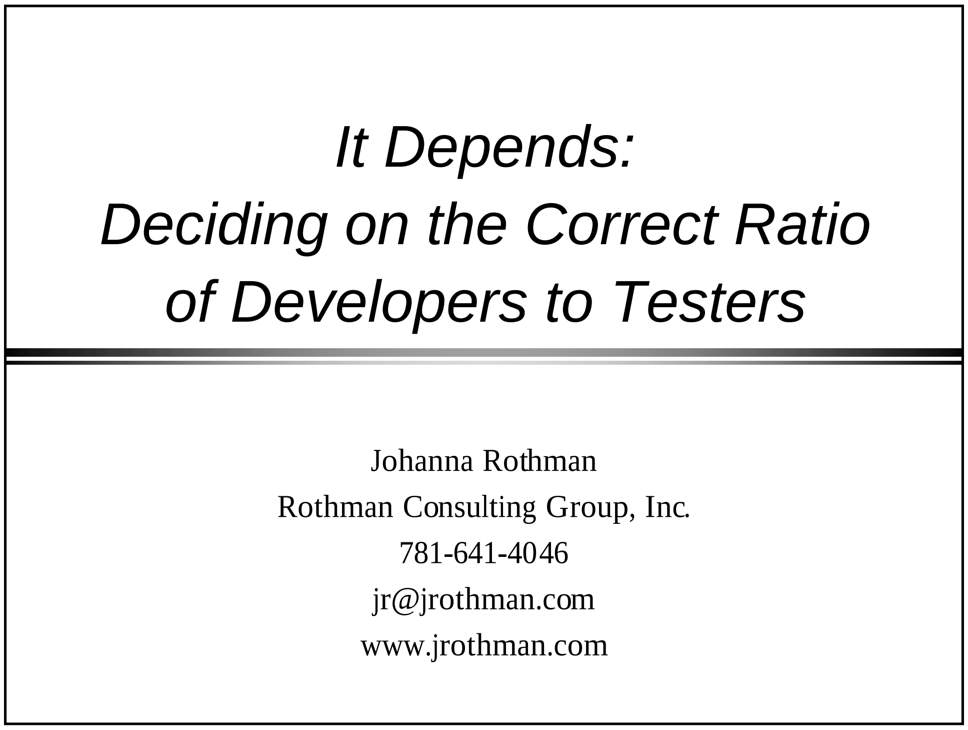
Thursday, May 4, 2000
1:30PM

It Depends: Deciding on the Correct Ratio of Developers to Testers

Johanna Rothman
Rothman Consulting Group, Inc.

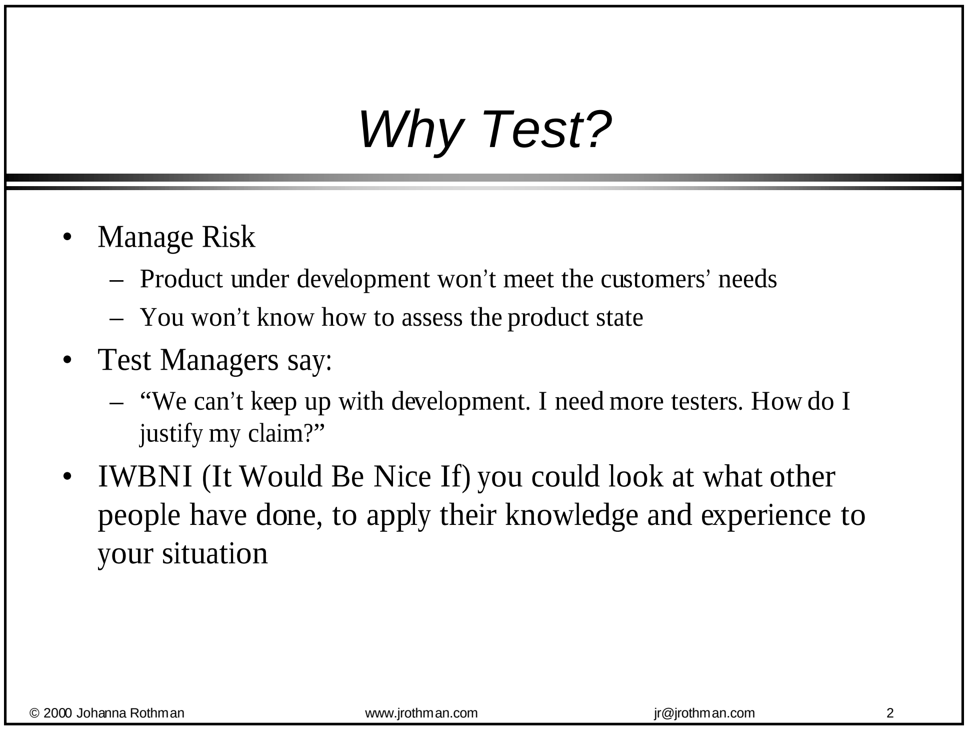
International Conference On
Software Testing Analysis & Review
May 1-5, 2000
Orlando. FL. USA

It Depends:
Choosing the Correct Developer to Tester Staffing Ratio



It Depends: Deciding on the Correct Ratio of Developers to Testers

Johanna Rothman
Rothman Consulting Group, Inc.
781-641-4046
jr@jrothman.com
www.jrothman.com



Why Test?

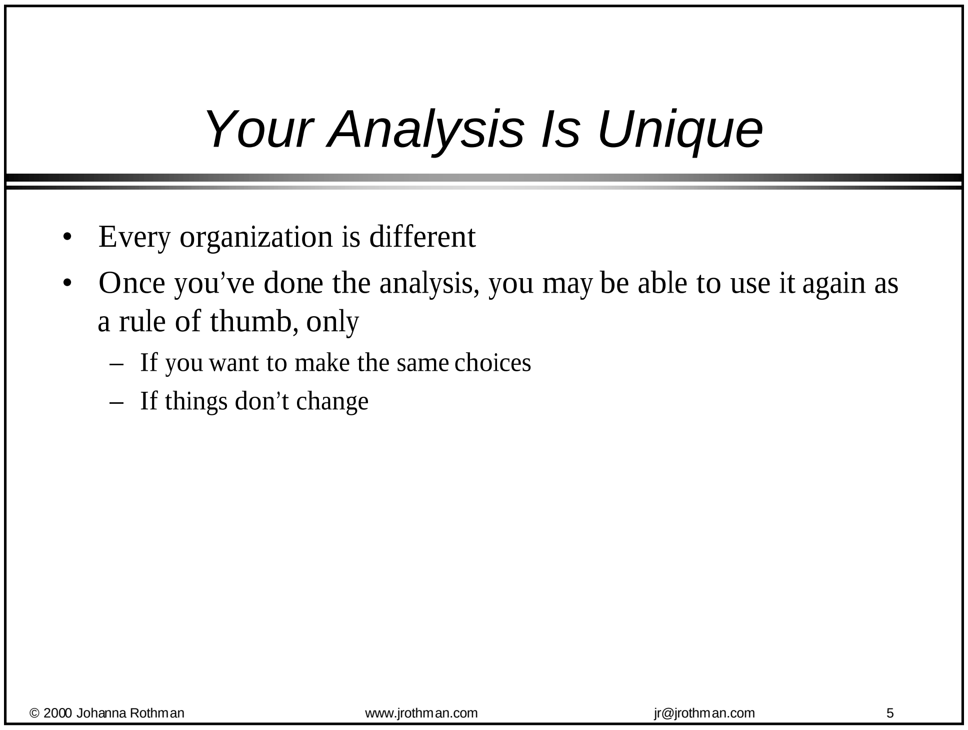
- Manage Risk
  - Product under development won’t meet the customers’ needs
  - You won’t know how to assess the product state
- Test Managers say:
  - “We can’t keep up with development. I need more testers. How do I justify my claim?”
- IWBNI (It Would Be Nice If) you could look at what other people have done, to apply their knowledge and experience to your situation

What We Know About Other Organizations

- Published developer:tester ratios of:
  - 1:1.5
  - 10:1
- Wide range variability implies that you have to perform some analysis of your situation

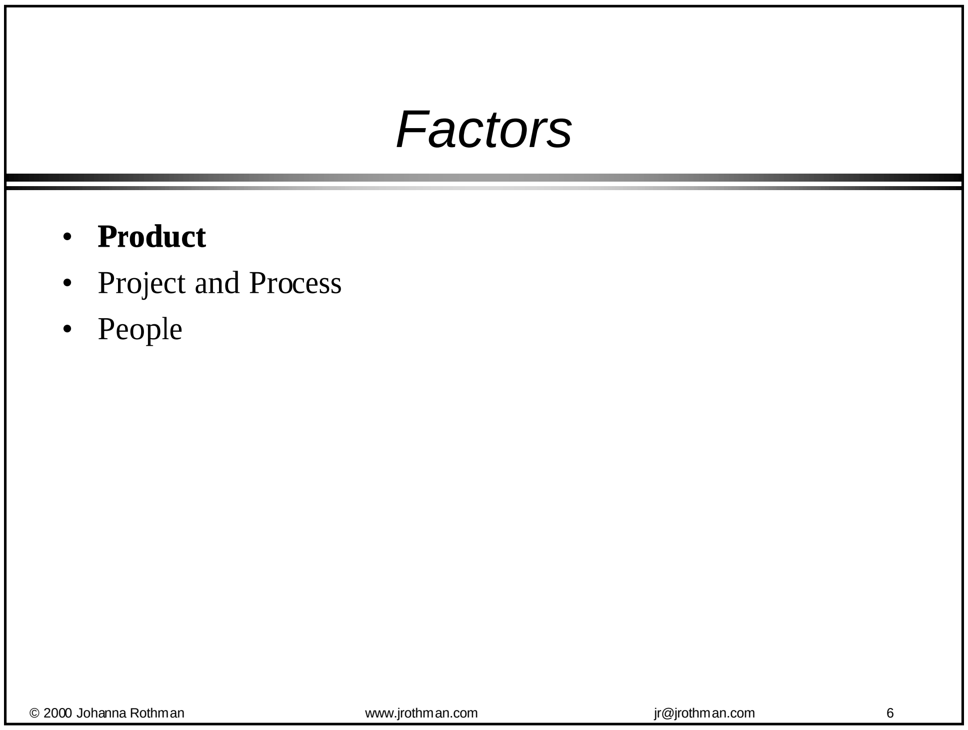
What Do You Need to Analyze?

- Your product
  - Requirements
  - Size
  - Complexity
- Your project and its process
  - Your customers' tolerance for defects and ship delays
  - How you develop products as an organization
- Your people
  - Development staff capabilities and responsibilities
  - Test staff capabilities and responsibilities



Your Analysis Is Unique

- Every organization is different
- Once you've done the analysis, you may be able to use it again as a rule of thumb, only
  - If you want to make the same choices
  - If things don't change



Factors

- **Product**
- Project and Process
- People

Product Risks and Complications in Three Projects

- BusyApp
  - Large, complex application
  - Cross-platform development: 4 DBs, 4 OSs
  - Created in a combination of 4GLs
  - Data-dependent GUI
- DataCrunch
  - Large, complex application
  - Cross-platform: 8 OSs, 4 DBs
  - Well-defined API
  - Data-dependent GUI
- LineChecker
  - Smaller than the other two applications, but over 100,000 SLOC
  - For one embedded platform, real-time data manipulation
  - Well-defined API and CLI

Contrast Initial Staffing Ratios

Project	Developers	Testers	Initial Ratio
BusyApp	10	2	5:1
DataCrunch	18	2	9:1
LineChecker	20	4	5:1

Compare Product Attributes

- X's refer to areas of product risk
- The more areas, the higher the product development and test risk
- More testers may help mitigate the risk

Attribute	BusyApp	DataCrunch	LineChecker
Large system	X	X	
Complex, real-time system	X	X	X
No well-defined API	X		
Data-dependent GUI	X	X	
Fit un-architected features into the existing product			

Factors

- Product
- **Project and Process**
- People

The Projects and Their Processes

- How the project is conducted is critical to knowing how many testers you will need
  - Lifecycle
  - Customers' tolerance for ship delay and defects
  - Which software engineering practices you perform

BusyApp

- Used a Code-and-Fix lifecycle
- Under pressure at the beginning, so they bypassed requirements and high level design
- Testers still testing previous versions of the project, so they started late
  - May have invoked Brooks' Law
- Weekly build consequences:
  - Developers forgot about the fixes they'd made more than a week ago
- Project Manager increased ratio to 2:1during the project so that untested product areas could be tested, and fixes verified

DataCrunch

- Spiral lifecycle
- Peer review of designs and some fixes
- Evaluated defects with a cross-functional team to assess the impact on customers
- Project Manager increased ratio to 3:1 so that untested product areas could be tested, and fixes verified

LineChecker

- Design-to-schedule lifecycle
- Defined and reviewed architecture and low-level designs
- Peer-reviewed all designs, code, unit tests
- Nightly builds
- Testers worked with developers
- No increase of ratio during the project

Compare Process and Project Attributes

Project Attribute	Busy App	Data Crunch	Line Checker
Define and inspect requirements			
Define and inspect the product architecture			✓
Design each component and inspect the design		✓	✓
Developers test code before each build		✓	✓
Plan testing at a sufficiently low enough level (exploratory testing, automate, test scenarios)		✓	✓
Have a test infrastructure for test development		✓	✓
Assess impact of open defects in a cross-functional team		✓	✓
Daily build and smoke test			✓
Low customer tolerance for ship delay	✓		✓
Low tolerance for customer-reported defects			✓

Process Considerations

- The more "good things", i.e. best practices, the fewer testers you will need
- The market you sell into determines the process (lifecycle and practices) you need

Factors

- Product
- Project and Process
- **People**

People Capabilities

- A rule of thumb:
  - *1 sufficiently bad developer = an arbitrarily high number of testers*
  - *1 sufficiently good developer + 0 sufficiently bad developers = an arbitrarily low number of testers*
- People vary widely in their capabilities
- Productivity does not always correlate directly with experience
- Many times, you get what you pay for

People Capabilities Across Three Projects

- BusyApp
  - Didn't spend too much on salaries, so they hired people with only 4GL experience, not software engineering experience (developers or testers)
    - Didn't know how to do high level design, or peer review, inspections
    - Developers didn't realize how much their organization of the product and how much testing they did would affect the testing
- DataCrunch
  - Peer-reviewed their code
  - Testers started off as developers, so they had an automation-only mindset
- LineChecker
  - Mentoring program for developers and testers
  - Considered when to do which kind of testing

Compare People Attributes Across Three Applications

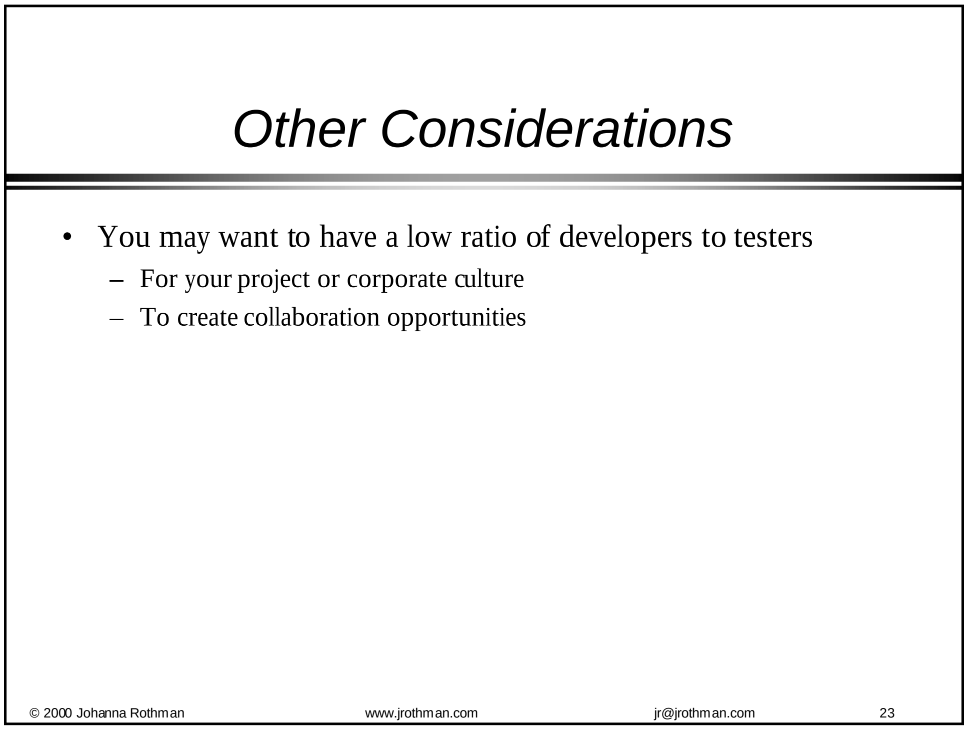
People factors	Busy App	Data Crunch	Line Checker
Developer experience in product domain	✓	✓	✓
General development team experience (perform inspections, understand designs)		✓	✓
Tester experience in product domain		✓	✓
General test team experience (perform inspections, understand designs, perform a large range of testing including exploratory and automated testing)			✓

A Little More About People

- Considerable variation of capability within organizations and across organizations
- The more capable the developers, the fewer testers needed
- The more capable the testers, the fewer testers needed
- You will need more testers
  - If the developers don't test their work from the beginning
  - If your testers can't test the product in depth
  - If your testers do only one kind of testing

Ratio Comparison at End of Projects

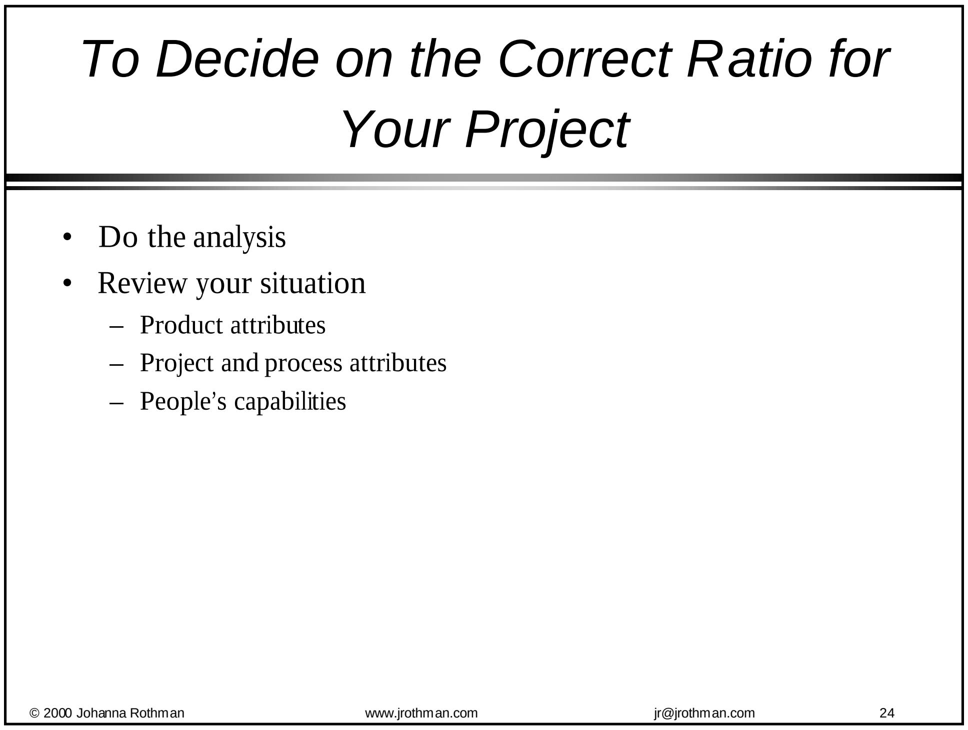
Project	Start Dev's	Start Testers	Start Ratio	End Dev's	End Testers	End Ratio	Successful Ship?
Busy App	10	2	5:1	14	7	2:1	No, too many defects found by customers
Data Crunch	18	2	9:1	18	6	3:1	No, very late
Line Checker	20	4	5:1	20	4	5:1	Yes



Other Considerations

- You may want to have a low ratio of developers to testers
  - For your project or corporate culture
  - To create collaboration opportunities

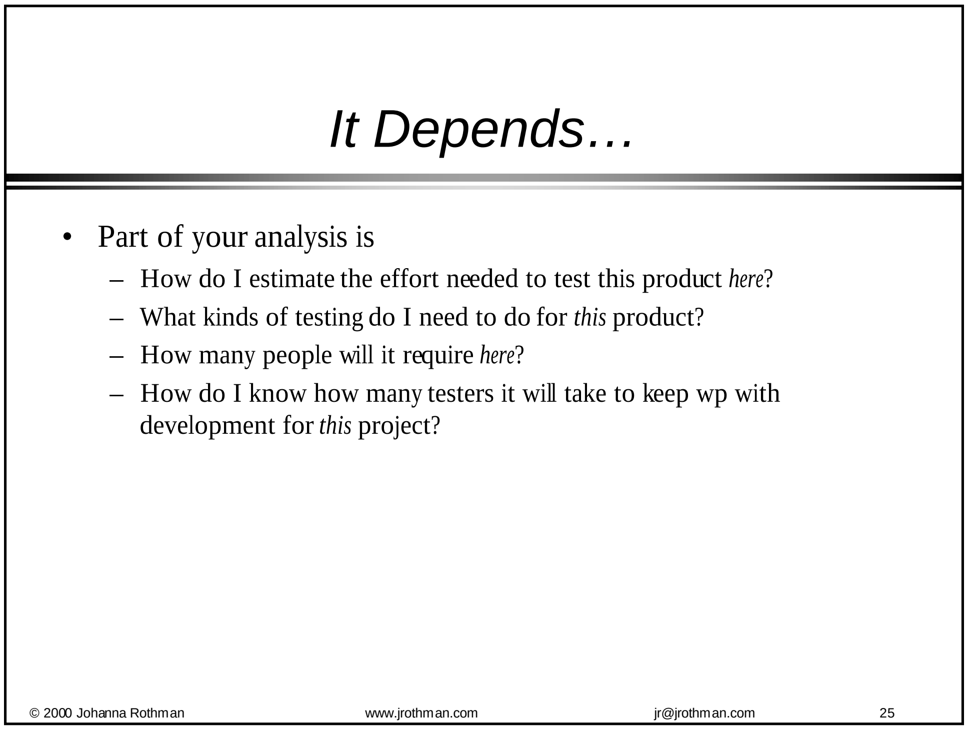
© 2000 Johanna Rothman www.jrothman.com jr@jrothman.com 23



To Decide on the Correct Ratio for Your Project

- Do the analysis
- Review your situation
  - Product attributes
  - Project and process attributes
  - People's capabilities

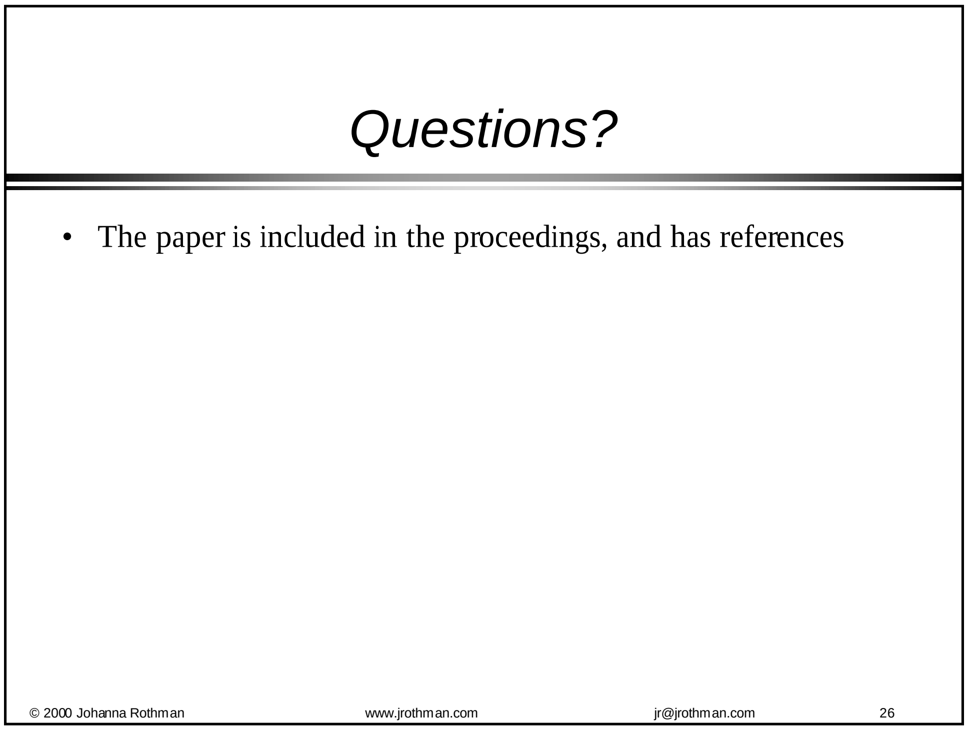
© 2000 Johanna Rothman www.jrothman.com jr@jrothman.com 24



It Depends…

- Part of your analysis is
  - How do I estimate the effort needed to test this product *here*?
  - What kinds of testing do I need to do for *this* product?
  - How many people will it require *here*?
  - How do I know how many testers it will take to keep wp with development for *this* project?

© 2000 Johanna Rothman www.jrothman.com jr@jrothman.com 25



Questions?

- The paper is included in the proceedings, and has references

© 2000 Johanna Rothman www.jrothman.com jr@jrothman.com 26

It Depends: Deciding on the Correct Ratio of Developers to Testers

© 2000 Johanna Rothman

Abstract

Many of us would like a precise answer to the question: “What’s the correct staffing ratio for developers to testers in my product development organization?” Usually though, the only answer is “It depends”. Your answer depends on your situation: the kind of project you’re working on, your schedule constraints, the culture you work in, and the quality expectations for the product. This paper discusses the thought process involved in deciding on your correct staffing ratios.

Introduction

Testing is about managing risk—the risk that the product you're developing somehow won't meet the customers' needs or the risk that you can’t get enough testing done to assess the product state. When these managers then say, “My testing staff can’t keep up with development. I need more testers. I need some kind of standards to justify my claim”, they're looking for ways to influence their senior managers. They would like to be able to point to some standard and say “Those people have our kinds of risks, and they have these many testers.”

There are some resources available, such as the comp.software.testing FAQ or discussions on the swtest-discuss mailing list, to see what other organizations use as development to test staff ratios. The developer to tester ratios range from 1:1.5 to 10:1, including organizations, such as Microsoft, which have ratios that are 1:1 [Cusamano]. With such a wide variability, you can't just choose someone else's ratios, you have to analyze your situation and derive an appropriate number of testers for your situation. That analysis needs to consider:

1. Your product (the requirements, product size, and complexity of the product)
2. Your project and its process (how your organization develops products, and your customers' tolerance for defects and ship delays)
3. Your people (development and test staff's abilities and responsibilities)

Since these issues are different for every organization, you and your organization have to analyze the choices you're making for products, process, and people. You may be able to apply the same analysis again, as a rule of thumb on future projects, if you continue to make the same choices, and if things don't change.

Product Risks/Complications

Let’s see how several sample projects selected their balance of development and testing team members.

BusyApp is a large, complex business application that enables software companies to manage product sales (which customers have bought how many of which product). It runs on four database versions, and at least four operating system versions. It's developed in a variety of 4GLs, such as PowerBuilder. It contains thousands of files and stored procedures, combined into 15 major modules. It's large. Its actual size is unknown, because the technical staff don't know how to calculate function points, and because the 4GL tools don't easily allow the technical staff to calculate the LOC (Lines of Code). It has hundreds of windows, many with data-dependent views.

At the beginning of the project, Ben, the BusyApp project manager staffed the project with 10 developers and 2 testers, a ratio of 5:1. During the project, the testers complained to the developers, saying they were having trouble knowing if they were testing the right things. They were sure there was more testing to do, but they didn't know how to find the areas to test.

BusyApp had a number of features that made it difficult to test: it did not have a well-defined API, and with the data-dependent windows, the testers had to create many different tests to generate appropriate GUI-based test suites. Ben decided he needed more testers to have confidence in the product.

In contrast, DataCrunch is also a large (about 1Million LOC) data manipulation application. It also runs on multiple platforms (8 OS's and 4 DBs). It has a well-defined, customer-accessible API; completely written in C. It too has hundreds of windows, many with data-dependent views.

Dan, the DataCrunch project manager, staffed his project with 18 developers and 2 testers, a 9:1 ratio. Because DataCrunch didn't have "enough" testers, in Dan's words, the developers peer-reviewed each other's code.

LineChecker is a real-time manufacturing line application. It has a well-defined API and Command Line Interface (CLI), and contains roughly 120,000 LOC, in C++. The product's complexity is in its real-time response on the manufacturing line and in the accuracy of the algorithms to do the real-time data manipulation.

Laurie, the LineChecker project manager, staffed the project with 20 developers and 4 testers, a 5:1 ratio. She was satisfied that she had enough people to test the project.

Here is the initial staffing for each project:

Project	Start Dev's	Start Testers	Start Ratio
BusyApp	10	2	5:1
DataCrunch	18	2	9:1
LineChecker	20	4	5:1

Table 1: Initial Staffing for Each Project

Here's how the three projects compared with their product attributes:

Product Risk Attribute	BusyApp	DataCrunch	LineChecker
Large system	X	X	
Complex, real-time system	X	X	X
No well-defined API	X		
Data-dependent GUI	X	X	
Fit un-architected features into the existing product			

Table 2: Comparison of each product’s risk attributes

The X's refer to areas of product risk that having more testers may be able to help mitigate.

The more X's, the more testers you will require, just based on the product. Link the X’s to testing practices, such as exploratory testing to decide where to plan the testing in detail, and then increase your staffing until you can accomplish the planned testing.

The product is not the only consideration when determining staffing ratios. The project and its process are also important.

Project and Process Issues

BusyApp used very few of the well known best software engineering and management practices [SPMN, McConnell], such as defining and inspecting requirements, architecture, and designs. BusyApp used a Code and Fix project lifecycle [McConnell], and the results showed. Ben was under pressure from the start of the project to deliver at a certain date, so the project team bypassed the requirements and the high-level design parts of the project. They were not able to perform nightly builds, due to the instability of the software. The testers were still busy on the previous project, so they could only start testing at the end of the project, and received builds about once a week to test. By the time the testers were ready to start testing a series of fixes, the developers had lost context and forgot the details of the previous week’s fixes.

Partway through system testing, Ben decided he needed additional developers and even more testers. By the end of the project, he had 14 developers and 7 testers, a 2:1 ratio. Ben maintained at least that ratio for the next three releases; to try to test away the damage caused by the original release.

DataCrunch used some good software engineering techniques; especially peer review of designs and fixes, and evaluating defects with a cross-functional team to assess their importance. Dan also decided to hire more testers and to extend the delivery date to meet their quality criteria for the release. Dan hired four more testers, to bring the ratio of developers to testers down to 3:1.

LineChecker used many good software engineering techniques. They did not define requirements thoroughly. The LineChecker project team thoroughly defined and reviewed their architecture. They also peer-reviewed all designs, code, and unit tests before creating a build for system test. The testers were encouraged to work with the developers as the developers wrote the code, so that the testers could plan their tests and their testing infrastructure.

This table contrasts the process and project attributes between the three applications:

Project Attribute	BusyApp	DataCrunch	LineChecker
Define and inspect requirements			
Define and inspect the product architecture			✓
Design each component and inspect the design		✓	✓
Developers test code before each build		✓	✓
Plan testing at a sufficiently low enough level (when to do exploratory testing, if and when to automate, what to test under which conditions)		✓	✓
Have a test infrastructure for easy test development		✓	✓
Review open defects in a cross-functional team to assess impact		✓	✓
Daily build and smoke test			✓
Low customer tolerance for ship delay[1]	✓		✓
Low tolerance for customer-reported defects			✓

Table 3: Contrast Process and Project Attributes

The BusyApp project team performed none of the software engineering practices. They depended on just the testers to test the product. Although they were under significant pressure to ship, it was not a serious problem (as in death of the company) for their customers to report defects.

DataCrunch performed more of these good practices, but not enough to maintain the original very high ratio of developers to testers. DataCrunch's releases were frequently very late, e.g. 9 months late for a 9-month release.

LineChecker did enough of these practices to maintain their ratio of developers to testers. Their customers could not tolerate high rates of defects, or even a defect bad enough to shut down the manufacturing line.

The more of these “good process” things you do, the fewer testers you will need. The fewer of these things you do, the more testers you will need.

Product and process issues are not the only issues to consider. The people working on the project are just as important as the other factors.

[1] If you want to read more about deciding which attribute of time to market, feature set, or low defects is most appropriate for your project, see “Using Quality to Drive Project Lifecycles”, http://www.jrothman.com/Papers/qualitydrivenlifecycles.html.

People and their Capabilities

The more capable your developers, the fewer testers you need. The more capable your testers, the fewer testers you need. If you hire developers who don't test their work, or if you hire all one kind of tester, and they can't test the product in depth, you will need more testers.

I had this conversation with a reviewer [Weinberg]. We came up with this rule of thumb:

> *1 sufficiently bad developer = an arbitrarily high number of testers*
>
> *1 sufficiently good developer + 0 sufficiently bad developers = an arbitrarily low number of testers*

Not only can one developer impact the number of testers needed, but also variation in abilities is a key issue for determining the number of testers needed. DeMarco and Lister [DeMarco] discovered significant variation in technical staff's capabilities during their coding war games.

Productivity does not always correlate positively with experience. Good developers and testers are not always the oldest ones, or the ones with the most experience.

There is considerable variation in people's abilities.

BusyApp's management team didn't want to spend too much on developer and tester salaries, so they hired people with 4GL experience, but not very much software *engineering* experience. They didn't know how to do high level design, or how to do peer review or inspections. They didn't understand how to best use the configuration management tool. The developers weren't stupid, they just didn't realize that if they organized the product better, and tested the product before the product got to the testing group, the project would be much shorter. The developers had experience with this kind of product; this wasn't their first release.

BusyApp's testers were smart people, all of whom were manual, GUI-based testers. They were good exploratory testers, but weren’t able to read code, and did not know how to use the automation tool to develop automated tests. When they received new builds, they would run the same manual tests as the last time, to verify that no bad fixes were inserted. They didn't understand the value of automating, or consider when to automate a test. They learned about the product on this project.

DataCrunch's developers were more sophisticated than BusyApp's. They looked for their buddies to peer-review their code. They understood the product, how it was used, and who used it, very well. DataCrunch's testers came from the development team. They were interested in testing, so they transferred over to testing. They had great development experience, but were lacking some basic test experience, such as when to do exploratory testing, and when to automate tests.

LineChecker's developers ranged from just out of school to more than 15 years of experience at the company. LineChecker had an explicit and active mentoring program. Each developer was paired with at least one other developer to do design and review. (This wasn't XP, Extreme Programming *a la* Beck, but was related to it.) Laurie, the

LineChecker project manager, scheduled reviews into the project schedule, so it was easy for the developers to do them, and hard for the developers to avoid doing.

LineChecker's testers had all been developers in a previous life. They were included in all the inspections. Their first approach to testing was to automate the tests, but they were able to conduct preliminary exploratory testing, and then to use exploratory testing when appropriate later in the project. They were able to develop their automated test infrastructure, and use it from the beginning of the project.

This table contrasts the people attributes between the three applications:

People factors	BusyApp	DataCrunch	LineChecker
Developer experience with this kind of product	✓	✓	✓
General development team experience (able to perform inspections, able to understand designs)		✓	✓
Tester experience with this kind of product		✓	✓
General test team experience (able to perform inspections, able to understand designs, able to perform a large range of testing, including exploratory and automated testing)			✓

Table 4: Contrast People Attributes

These three projects all wanted to know the optimal ratio of developers to testers, and they've all come up with independent conclusions.

The three projects had a planned number of developers and testers. Here is the initial, final staffing, and results for each project:

Project	Start Dev's	Start Testers	Start Ratio	End Dev's	End Testers	End Ratio	Successful Ship?
BusyApp	10	2	5:1	14	7	2:1	No, too many defects found by customers?
DataCrunch	18	2	9:1	18	6	3:1	No, very late
LineChecker	20	4	5:1	20	4	5:1	yes

Table 5: Initial and Final Staffing Ratios, and Results per project

As you can see, there is no one right ratio. BusyApp and LineChecker both had the same ratio of developers to testers at the start, but because their products were different, their projects and processes were different, and they had a different caliber of people on the project, the needs were different. The same ratios did not work for the different projects.

BusyApp is now using a 1:1 ratio, and is changing how they work (their process). DataCrunch is maintaining the 3:1 ratio, and has changed how they work, so that they can release on time. LineChecker has maintained the 5:1 ratio over the lifetime of the product. Laurie, the project manager, recently started a new project for a new product, and decided to use a ratio of 4:1 for the new project, since the customers have a lower tolerance for defects, and the product contains a GUI.

Culture

There are other reasons to have a higher or lower developer to tester ratio, not based on purely analytical reasoning, but based more on the culture you want to create. For example, you may want to encourage a collaborative product development environment with a 1:1 ratio, so that developers and testers become buddies.

Summary

So, what's a project manager or a test manager to do? You can reframe the test ratio question into a series of questions you may be able to answer to help you decide how many testers you need.

4. How do I estimate the effort needed to test this product *here*?
5. What kinds of testing do I need to do *for this product*?
6. How many people will it take to do that work *here*?
7. How do I know how many testers it will take to keep up with development *for this project*?

Review your product attributes. Add up your X's, and see how risky your project is. Then review how your team develops the project. Can you check off any of the software engineering practices? If so, that reduces the risk of not enough testing.

Then look at the pressures on your project. Do you have substantial pressure to ship by a certain date, or are your customers intolerant of shipped defects? That increases the risk, and implies you should have more testers.

Then review the team's capabilities on the project. Do your developers know how to create product in a low-defect way? Do they understand the product? How flexible are your testers? Can they change how they test according to where in the project they are, and the kind of product they're testing? If you can't check off any of the people attributes here, you have substantial risk, and will need more people.

When you're looking for a standard answer, consider whether there really is One Right Answer. Analyze the problem before you can give an answer. It all depends—and what it really depends on is the analysis you perform of the risks and tradeoffs.

Acknowledgements

I thank the reviewers for their helpful comments: James Bach, Mark Druy, Danny Faught, Bret Pettichord, Jerry Weinberg, Mark Weisz, Karl Wiegers, Mark Wiley.

References

1. Beck, Kent, Extreme Programming, Addison Wesley, Reading MA. 2000.

2. Cusamano, Michael A. and Richard W. Selby, Microsoft Secrets, Simon and Schuster, Inc., New York, 1995.
3. DeMarco, Tom, and Tim Lister, Peopleware, Productive Projects and Teams, second edition, Dorset House Publishing, New York, 1999.
4. Software Program Manager's Network, *Best Practices,* 1999, on www.spmn.com
5. McConnell, Steve. Rapid Development, Microsoft Press, Redmond WA, 1996.
6. Weinberg, Gerald M. Private email communication, Feb. 2000.

Johanna Rothman

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles *in Software Development*, *Cutter IT*, *IEEE Computer*, *Software Testing and Quality Engineering*, *and IEEE Software*.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.